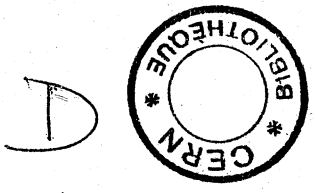


C.R.N.

centre de recherches nucléaires de Strasbourg

CRN/HE 78-28

THESE

présentée

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE

par

Gerard HUOT

CONCEPTION ET REALISATION D'UN SYSTEME DE DEVELOPPEMENT, EXECUTABLE SUR MINI-ORDINATEUR NORD-10, POUR UN MICROPROCESSEUR SPECIALISE

Groupe Compteurs (PNHE)

Institut National
de Physique Nucléaire
et de Physique
des Particules

Université
Louis Pasteur
de Strasbourg

67037 STRASBOURG CEDEX FRANCE

THESE

CRN/HE 78-28

présentée

A L'UNIVERSITE LOUIS PASTEUR DE STRASBOURG

pour obtenir le grade de

DOCTEUR DE SPECIALITE
Option: Electronique et Instrumentation

par

Gerard HUOT

CONCEPTION ET REALISATION D'UN SYSTEME DE DEVELOPPEMENT, EXECUTABLE SUR MINI-ORDINATEUR NORD-10, POUR UN MICROPROCESSEUR SPECIALISE

Soutenue le 6 Decembre 1978 devant la Commission d'Examen:

MM.	*G.SUTTER*	*President*
	G.METZGER	*Examinateurs*
	J.M.MEYER	
	Y.CHATELUS	

UNIVERSITÉ LOUIS PASTEUR
STRASBOURG I

Edition 1978

LISTE DES PROFESSEURS, MAITRES DE CONFÉRENCES DIRECTEURS ET MAITRES DE RECHERCHES CNRS ET INSERM

Président	Professeur P.KARLI
Vice-Présidents	Professeur Fr.MARCOUX Professeur J.H.WEIL
Président Honoraire	Professeur G.OURISSON
Secretaire Général	Monsieur G.KIEHL

U.E.R. DE SCIENCES EXACTES

U.E.R. de Mathématiques	Directeur	Jacques FARAUT
U.E.R. de Sciences Physiques et Chimiques	Directeur	Jean DEHAND
U.E.R. de Sciences de la Matière	Directeur	Jean José FRIED
U.E.R. de Sciences de la Vie et de la Terre	Directeur	Yves BOULANGER
U.E.R. de Sciences du Comportement et de l'Environement	Directeur	Michel TARDY
U.E.R. de l'Ecole d'Application des Hauts Polymères	Directeur	Constant WIPPLER
U.E.R. de l'Ecole Nationale Supérieure de Chimie	Directeur	Marc DAIRE
U.E.R. de l'Observatoire	Directeur	Alphonse FLORSCH
U.E.R. de Physique du Globe	Directeur	Alexandre ROCHE

DOYENS HONORAIRES : P.LACROUTE – J.H.VIVIEN – G.MILLOT.

PROFESSEURS HONORAIRES : L.BOISSELET – J.BYE – H.CARTAN – G.CERF – C.CHABAUTY – A.CHRETIEN – J.DENY – Ch.EHRESMANN – Mlle S.GILLET – S.GOLDSZTAUB – R.HOCART – P.LACROUTE – G.LEMEE – P.L'HERITIER – A.LICHNEROWICZ – A.MAILLARD – L.NEEL – J.PARROD – R.ROHMER – J.P.ROTHE – L.SACKMANN – Ch.SADRON – H.SAUCIER – H.VILLAT – H.WEISS – Et.WOLFF – J.YVON.

MAITRE DE CONFÉRENCES HONORAIRE : R.WEIL.

PROFESSEURS

J.P. ADLOFF	Chimie Nucléaire
R. ARMBRUSTER	Physique
V. AVANISSIAN	Analyse supérieure
F. BECKER	Physique Mathématique
Cl. BENEZRA	Dermato-Chimie
H. BENOIT	Physicochim. macromol.
P. BENVENISTE	Physiologie Végétale
D. BERNARD	Méth. Math.de la Phys.
J.C BERNIER	Chimie Générale
Y. BOULANGER	Chimie Biologique
J. BRENET	Electrochimie
M. BRINI	Chimie
J. BROSSAS	Chimie Macromol.
R. CERF	Physique générale
P. CHEVALLIER	Physique
A. CLAUSS	Chimie
A. COCHE	Physique Nucléaire
M. DAIRE	Chim.Phys.Ind.et Sc.des Matér.
H. DANAN	Phys.Atom.et Phys.du Solide
E. DANIEL	Physique Expérimentale
M. DAUNE	Biophysique
J. DEHAND	Chimie Minérale
A. DELUZARCHE	Chimie
G. DUNOYER de SEGONZAC	Géologie
H. DURANTON	Botanique
J.P. EBEL	Chimie Biologique
J.P. EBERHART	Minéralogie
J. FARAUT	Mathématiques
P. FEDERLIN	Chimie
X. FERNIQUE	Mathématiques
D. FOATA (dét.)	Mathématiques
E. FOLLENIUS	Zoologie
J.J. FRIED	Méc. des Fluides
D. FROELICH	Chim.Gén.Chim.Phys.
A. FUCHS	Mécanique rationnelle
A. GAGNIEU	Botanique
A. GALLMANN	Physique
F. GAULT	Chimie Organique
F. GAUTIER	Physique
R. GERARD	Mathématiques
G. GLAESER	Mathématiques
O. GODBILLON	Mathématiques
S. GORODETZKY	Phys.Gén.et Phys.Nucl.
M. GOUNOT	Botanique
M. GROSS	Chimie
M. GROSMANN	Physique
L. HIRTH	Microbiologie
P. JOLY	Biologie Générale
J.P. JOUANOLOU	Mathématiques
T. JUTEAU	Minéral.et Pétrographie
R. KIRSCH	Zoologie
F. LACROUTE	Biologie Végétale
R. LECOLAZET	Physique du Globe
J.M.LEHN	Chimie
J. LEITE-LOPES	Phys.Nucl.et Corpuscul.
M. LEROY	Chimie
J. LUCAS	Géologie
D. MAGNAC	Physique
J. MARTINET	Mathématiques
P. MIALHE	Physiologie Animale
A. MICHARD	Géologie
M. MIGNOTTE	Informatique
G. MILLOT	Géologie et Paléontologie
G. MONSONEGO	Physique Théorique
B. MORIN	Mathématiques I
G. OURISSON	Chimie
J.P. RAMIS	Mathématiques Générales
G. REEB	Topologie
Ph. RICHARD	Physiologie Animale
J.J. RIEHL	Chimie
Cl. ROBERT	Physique
A. ROCHE	Physique du Globe
Ph. ROPARTZ	Psycho-Physiologie
J. ROUX	Botanique
Fr. SCHALLER	Biologie Générale
G. SCHIFFMANN	Mathématiques
J.P. SCHWING	Chimie
M.J. SCHWING	Chimie Physique
M. SIESKIND	Physique
G. SOLLADIE	Chimie Organique
J. SOMMER	Chimie Appliquée
F. STUTINSKY	Physiologie Générale
G. SUTTER	Physique Electronique
Ch. TANIELIAN	Chimie
J.J. THIEBOLD	Biologie Animale
D. VIAUD	Mathématiques
J.H. VIVIEN	Zool.et Embryol.Expérim.
R. VOLTZ	Physique Théorique
J.H. WEIL	Chimie Biologique
G. WEILL	Physique
R. WEISS	Chimie
P.L. WENDEL	Physique
C. WIPPLER	Physicochim.des Hts Polym.Indus.
J. WUCHER	Physique
B. WURTZ	Chimie Biologique

PROFESSEURS ASSOCIÉS

A. ANDREOTTI	Mathématiques
A. BANDERET	E.A.H.P.
B. BOURROUILH	Géophysique
T. EGGARTER	Physique
V. ERN	Physique
G.L.HALLER	Chimie
T. HIRAI	Mathématiques
C. JASCHEK	Astronomie
J. OSBORN	Chimie Minérale
J. SMID	Physique

PROFESSEURS CONVENTIONNÉS ULP : P.BOUVEROT (Physiol.Respiratoire) – P.DEJOURS (Physiol.Respiratoire).

MAITRES DE CONFÉRENCES :

J.Fr. BOUTOT	Mathématiques	
Ch. BURGGRAF	Minéralogie	
H. BURNAGE	Mécan.des Fluides	
T. JUTEAU	Minéralogie	
J.Cl. LAFON	Informat.Appliquée	

ASTRONOME ADJOINT : A.FLORSCH (Astronomie).

MAITRE DE CONFÉRENCES ADJOINT : J.SITTLER (Géologie).

MAITRES DE CONFÉRENCES ASSOCIÉS : C.DADE (Mathématiques) – Y.NAKATANI (Chimie) – H.UMEMURA (Mathématiques).

DIRECTEURS DE RECHERCHE C.N.R.S.

J.F. BIELLMANN	Chimie	J. MARCHAL	Physicochim.Macromolécul.	A. PORTE	Biologie Cellulaire
P. BOUVEROT	Physiol.Respiratoire	PA.MEYER	Mathématiques	P. REMPP	Physicochim.Macromol.
P. DEJOURS	Physiol.Respiratoire	A.J.P.MEYER	Physique	A. SKOULIOS	Physicochim.Macromol.
A. KNIPPER	Phys.Nucl.et Corpusc.	J. MEYER	Botanique	A. VEILLARD	Chimie Moléculaire
A. KOVACS	Physicochim.Macromol.	C. MIALHE	Physiologie	A. ZUKER	Physique Théorique

MAITRES DE RECHERCHE C.N.R.S.

J.Ch. ABBE	Physicoch.atomique et ionique chimie nucléaire	
P. ALBRECHT	Chimie	
F. BECK	Physique Nucléaire	
G. BECK	Biochimie	
J.P. BECK	Physiologie	
R. BERTINI	Physique Nucléaire	
M. BONHOMME	Géologie	
H. BRAUN	Physique Corpusculaire	
M.C. CADEVILLE	Physique des Solides	
H. CALLOT	Chimie	
S. CANDAU	Physique	
M. CHAMPAGNE	Biophysique	
M. CHOUDHURY	Physique du Globe	
J.P. COFFIN	Phys.Nucléaire et Corpusculaire	
A. CORET	Physique	
M. CROISSIAUX	Physique Nucléaire	
Cl. DELLACHERIE	Mathématiques	
D. DISDIER	Physique Nucléaire	
J. DOUBINGER	Géologie	
S. EL KOMOSS	Physique	
M. FRANCK-NEUMANN	Chimie Organique	
E. FRANTA	Physicochimie Moléculaire	
J.M. FRIEDT	Physicoch.atomique et ionique	
A. FRIDMANN	Physique Corpusculaire	
Y. GALLOT	Physicochimie Macromoléculaire	
Ph. GRAMAIN	Physicochimie Macromoléculaire	
J.B. GRUN	Physique	
J. HERZ	Physicochimie Macromoléculaire	
J. HOFFMANN	Biologie Animale	
G. KAUFMANN	Chimie	
B. KOCH	Physiologie	
P. LAURENT	Physiol.comparée des régulations	
Cl. LERAY	Physiol.comparée des régulations	
A. LLORET	Physique Corpusculaire	
B. LOTZ	Physicochimie Macromoléculaire	
G. MAIRE	Chimie	
A. MALAN	Physiologie Respiratoire	
E. MARCHAL	Physicoch.Moléculaire et Macromoléculaire	
R. MORAND	Physique Nucléaire	
Th. MULLER	Physique	
G. MUNSCHY	Physique	
M. NAUCIEL-BLOCH	Physique des Solides	
A. NICOLAIEFF	Virologie Végétale	
M. PATY	Physique Corpusculaire	
R. PFIRSCH	Botanique	
Cl. PICOT	Physicochimie Macromoléculaire	
P. POIX	Chimie	
J. POUYET	Biophysique	
B. REES	Chimie	
P. REMY	Biochimie	
J. RINGEISSEN	Physique	
J.P. ROTH(s/Dir.Inst.de Rech.)	Physicochimie Macromoléculaire	
F. SCHEIBLING	Spectrométrie Nucléaire	
N. SCHULZ	Physique Nucléaire	
C. SCHWAB	Physique	
R. SELTZ	Physique Nucléaire	
P. SIFFERT	Physique des Rayon.et Electronique Nucléaire	
Cl. SITTLER	Géologie	
Cl. STRAZIELLE	Physicochimie Macromoléculaire	
M. SUFFERT	Basses Energies	
K. TRAORE	Physicochimie atomique et ionique	
R. VAROQUI	Physicochimie Macromoléculaire	
G. WALTER	Physique Nucléaire	
Fr. WEBER	Géologie	
J.P. WENIGER	Zoologie	
J. WITZ	Biologie Cellulaire	
R. WOLFF	Chimie	
R. ZANA	Physicochimie Macromoléculaire	
J.P. ZILLINGER	Physique	

A MES PARENTS,

A CHRISTINE.

TABLE DES MATIERES

INTRODUCTION

Le groupe "Compteurs" du Laboratoire PNHE du Centre de Recherches Nucléaires (C.R.N.) de Strasbourg-Cronenbourg, travaille sur une expérience de physique des Hautes Energies auprès de l'Accélérateur SPS de 300 GeV au Centre Européen de Recherche Nucléaire (CERN).

Cette expérience a pour but d'étudier les désintégrations leptoniques d'hypérons chargés, principalement les Ξ^- et Σ^-. Le taux d'acquisition élevé jusqu'à 150 événements d'environ 1000 mots par burst, susceptibles d'être filtrés par programme et la complexité de l'appareillage expérimental a amené le groupe à développer un système d'acquisition à structure multiprocesseurs. Ce système spécialisé à l'acquisition et au filtrage des événements peut être interconnecté à n'importe quel ordinateur assurant le contrôle de l'expérience par le dialogue avec les physiciens.

Le système F.A.S. [1] (Fast Acquisition System) comporte un processeur spécialisé à l'acquisition de données : GESPRO [2].

Le Logiciel associé à ce micro-ordinateur est élaboré à l'aide d'un cross assembleur. Compte tenu du caractère évolutif de toute expérience de physique la complexité du logiciel a évolué en parallèle avec celle de l'expérience et l'aide apportée par cet assembleur s'est avérée insuffisante.

Il a donc été décidé de le remplacer par un système de développement beaucoup plus élaboré et plus performant; c'est ce qui a fait l'objet de notre travail.

Ce système a été conçu pour prendre en charge la programmation depuis l'écriture des programmes en langage symbolique à partir des jeux d'instructions jusqu'à leur exécution correcte.

CHAPITRE I

ENVIRONNEMENT ELECTRONIQUE-INFORMATIQUE ASSOCIE A L'EXPERIENCE HYPERON 300

I. INTRODUCTION

L'environnement électronique aux différents détecteurs de particules est composé de :

- l'électronique de lecture
- l'électronique d'acquisition

L'ensemble est constitué d'éléments modulaires interconnectés en standard CAMAC [3].

L'électronique d'acquisition étant la partie de l'expérience dans le cadre de laquelle a été conçu le système de développement relatif à GESPRO.

II. SYSTEME D'ACQUISITION F.A.S.

1. Généralités :

Le système d'acquisition est le système F.A.S.

Il a été développé, pour permettre l'acquisition d'au moins 100 événements de 800 mots chacun, pendant le temps utile du faisceau (burst) soit 1,4 secondes environ. Ce taux élevé d'événements acquis est assuré grâce à une structure multiprocesseurs du système. Il est constitué d'un mini-ordinateur NORD-10 et d'un microordinateur GESPRO, intégré au niveau

des entrées/sorties CAMAC de NORD. L'ensemble pouvant travailler seul ou en liaison avec un mini-ordinateur maître : DDP-516. (fig. I.1.)

Les différents châssis CAMAC, supportant les modules associés aux détecteurs, sont interconnectés selon une structure multibranche. L'interfaçage à NORD-10 est assuré par un contrôleur de système (E.S.C. - fig. I.2). C'est un châssis CAMAC muni d'un module d'entrées/sorties programmées (P.I.O.) et d'un module d'accès direct mémoire (D.M.A.).

L'accès CAMAC est multiplexé au niveau de l'E.S.C. entre NORD 10 et GESPRO permettant un accès indépendant à chacun d'eux (figure I.3).

2. Principe de fonctionnement :

Pendant le burst, GESPRO assure l'acquisition d'événements par l'intermédiaire d'un buffer de sa mémoire et effectue un prétraitement qui est en fait un filtrage logiciel.

Les événements sont transférés en mémoire NORD sélectionnés ou écrasés s'ils n'ont pas été retenus.

La mémoire de NORD est multiaccès pour permettre un accès direct de la part de GESPRO. La partie réservée aux données est constituée de 3 blocs de 4 K mots chacun. Chaque bloc est affecté par rotation à l'acquisition, au traitement et au transfert sur bande magnétique (voir fig. I.3).

Hors burst, GESPRO a la possibilité de faire des tests sur les modules CAMAC. Pendant ce temps NORD-10 effectue le traitement des données par l'intermédiaire d'une mémoire d'accumulation constituée d'un tambour magnétique et le transfert sur bande magnétique.

Le dialogue de synchronisation NORD-GESPRO est effectué par un jeu d'interruptions réciproques.

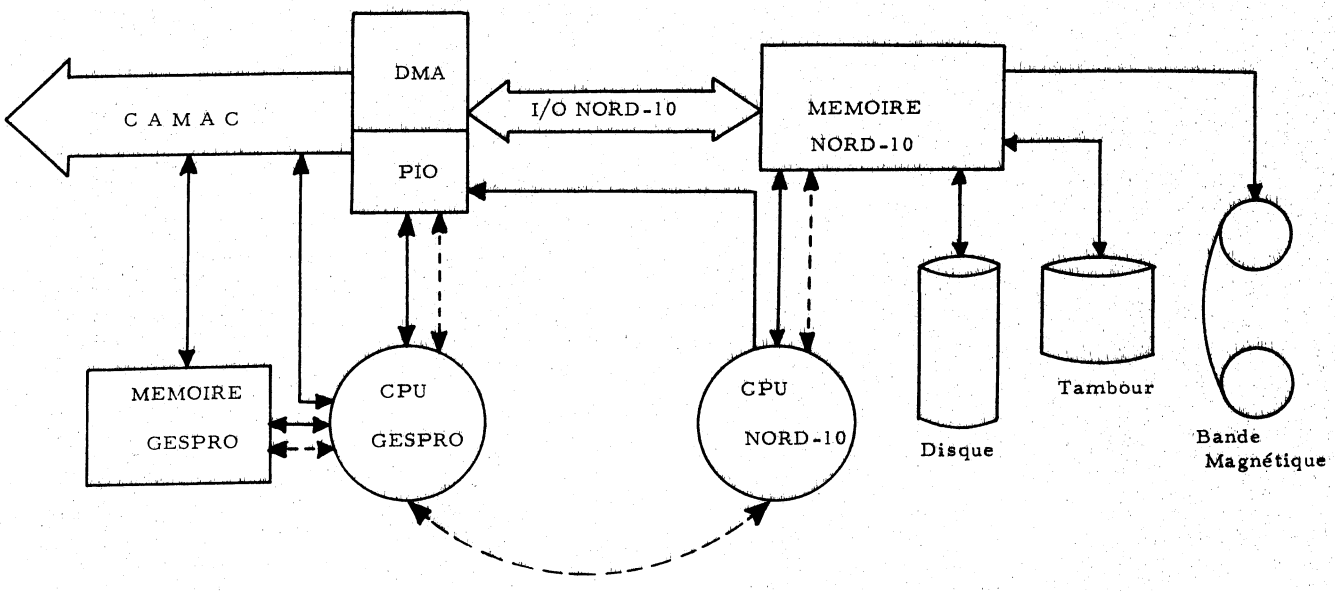


Fig. I. 1 : Structure du système d'acquisition F.A.S.

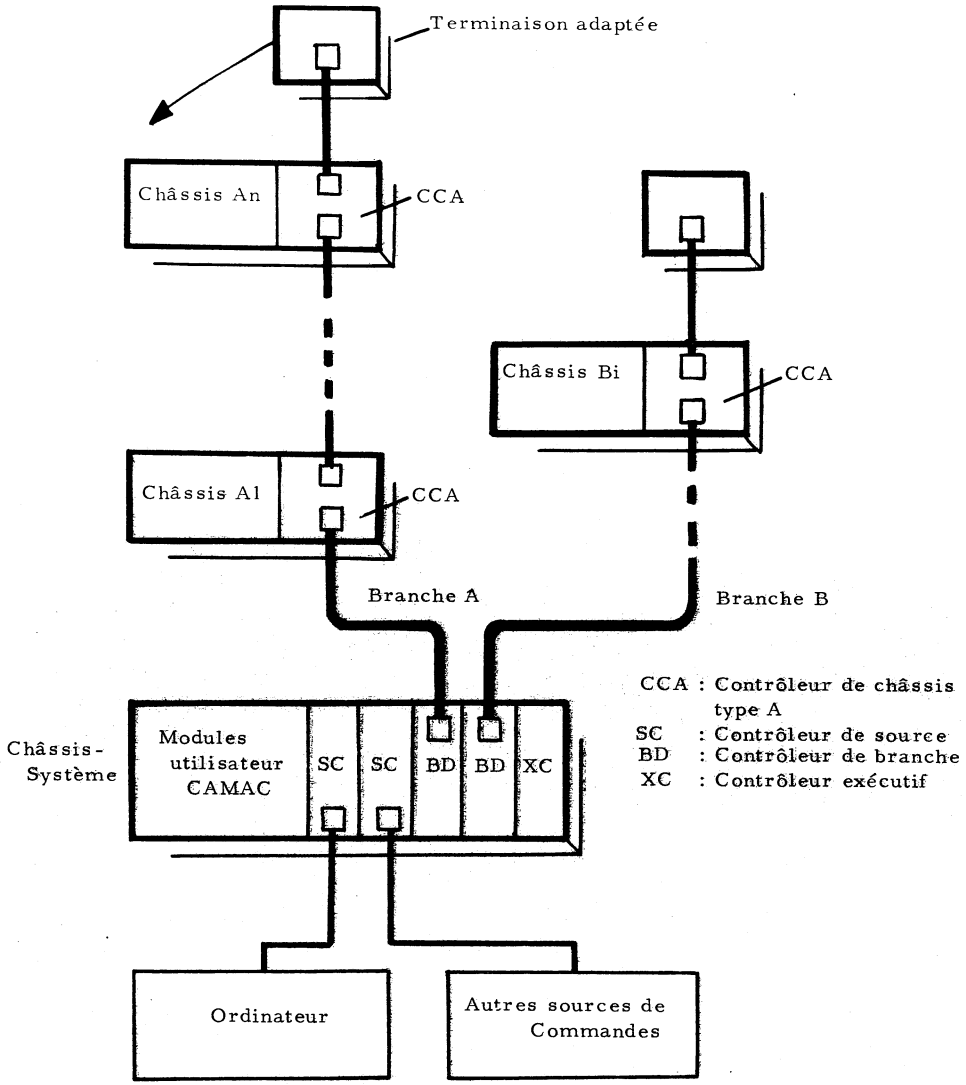


Fig.I.2 : Utilisation d'un châssis-système dans un système CAMAC multi-branche.

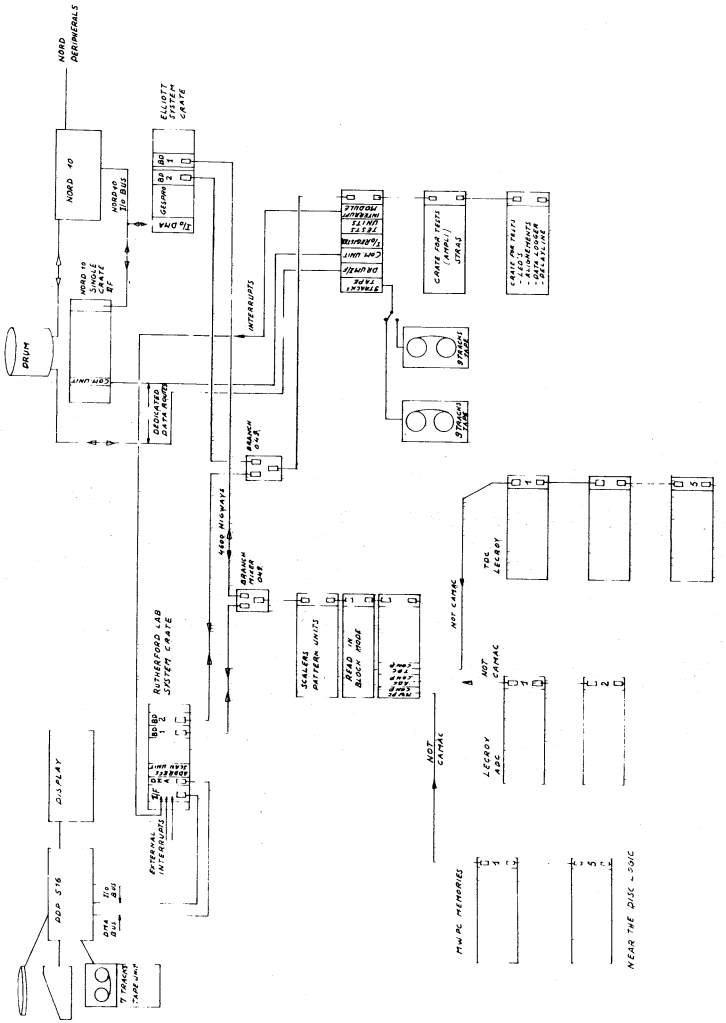


Fig.I.3

3. Le Microordinateur GESPRO :

3.1. Caractéristiques :

GESPRO est un processeur à mots de 24 bits, microprogrammable et réalisé à partir de "Bit Slice" INTEL 3000, avec un temps de cycle de 200 ns.

Il dispose d'une mémoire centrale d'une capacité mesurable de 32 K mots de 24 bits et d'une mémoire de microinstructions de 2K mots de 48 bits. Il a une structure d'entrée/sortie adaptée CAMAC, seul périphérique du processeur.

La figure I-4 donne la structure "hardware" du microprocesseur. Il peut être contrôlé à partir de NORD à l'aide d'une série d'ordres CAMAC où GESPRO est considéré comme périphérique de NORD-10.

3.2. Microprogrammation :

La microprogrammation du processeur permet en premier lieu d'adapter l'acquisition à l'évolution de l'expérience de physique, par la création de nouvelles instructions au fur et à mesure des besoins.

D'autre part la microprogrammation de tâches spécialisées permet de remplacer la programmation, quand des temps d'exécution extrêmement courts sont nécessaires.

Ce caractère particulier de GESPRO va intervenir sur la structure des instructions et sur l'élaboration du logiciel associé.

3.3. Structure des instructions GESPRO :

En raison de leur structure complexe la plupart des instructions occupent plusieurs mots de 24 bits.

Les champs sont spécialisés et donc différents d'une instruction à l'autre.

La figure I.5 a indique les différents modes d'adressage utilisés. La figure I.5.b. donne un exemple d'instruction spécialisée.

Les instructions de GESPRO peuvent se classer en deux groupes :

- Les instructions classiques :
  - référence mémoire
  - opérations arithmétiques et logiques
  - débranchements

  Elles sont implantées en mémoire de microinstructions de type ROM
- Les instructions spécialisées :
  - spécialisées système
  - spécialisées utilisateur

Le premier type se caractérise par leur utilisation répétitive dans un cadre toujours identique.

Les instructions spécialisées utilisateurs, varient avec certaines exigences expérimentales. Leur implantation est faite en mémoire de microinstructions de type R.A.M. et sont chargées en fonction des besoins. C'est en fonction de ces jeux d'instructions que le logiciel de GESPRO va être élaboré.

La spécificité des instructions d'une part et la constante évolution du jeu d'instructions vont caractériser de façon précise le système de développement.

Fig.I.5.a : Structure des instructions et modes d'adressage

TDC STR composée de 5 mots de 24 bits

adresse de débranchement suivant la façon dont se passe le traitement

- adresse table
- test si Q = 0
- test si overflow
- test si adresse décroissante
- test si - 1 suivi de 0

Fig.I.5.b : Exemple d'instruction spécialisée

Fig.I.4.

STRUCTURE DU PROCESSEUR CAMAC MICROPROGRAMMABLE.

4. Logiciel associé au système F.A.S. :

Le logiciel F.A.S. est modulaire, de façon à s'adapter aux différents stades de développement de l'expérience.

Les modules associés au traitement et dialogue avec l'opérateur sont des outils destinés à la physique donc écrits en Fortran.

Par contre les modules associés à l'acquisition et au transfert de données sont en assembleur, pour obtenir des temps morts d'acquisition et d'allocation mémoire aussi faibles que possible. Le traitement spécifique à GESPRO est microprogrammé.

4.1. Systèmes opérationnels spécialisés :

Le système spécialisé NORD10 assure

. la gestion des blocs mémoire NORD

. la gestion du tambour (données + histogramme)

. la synchronisation des tâches

Le système opérationnel associé à GESPRO assure la synchronisation des tâches avec NORD-10 et l'expérience.

4.2. Procédure de programmation :

Le développement de programmes est effectué sous SINTRAN III (Système d'exécution de NORD 10) disposant des facilités d'édition et de compilation.

Pour les algorithmes répartis sur les 2 machines, la partie relative à chacune est assemblée ou compilée sous son propre système c'est à dire SINTRAN III pour NORD 10 et un assembleur chargeur exécuté par NORD 10 pour GESPRO (Fig.I.6).

4.3. Cross-assembleur GESPRO

L'assembleur GESPRO a été conçu pour faciliter la tâche du programmeur.

C'est un assembleur exécuté en un passage sur NORD-10 sous SINTRAN III.

Il accepte de petits programmes source (200 instructions maximum) en langage symbolique et les traduit sous forme de programme objet

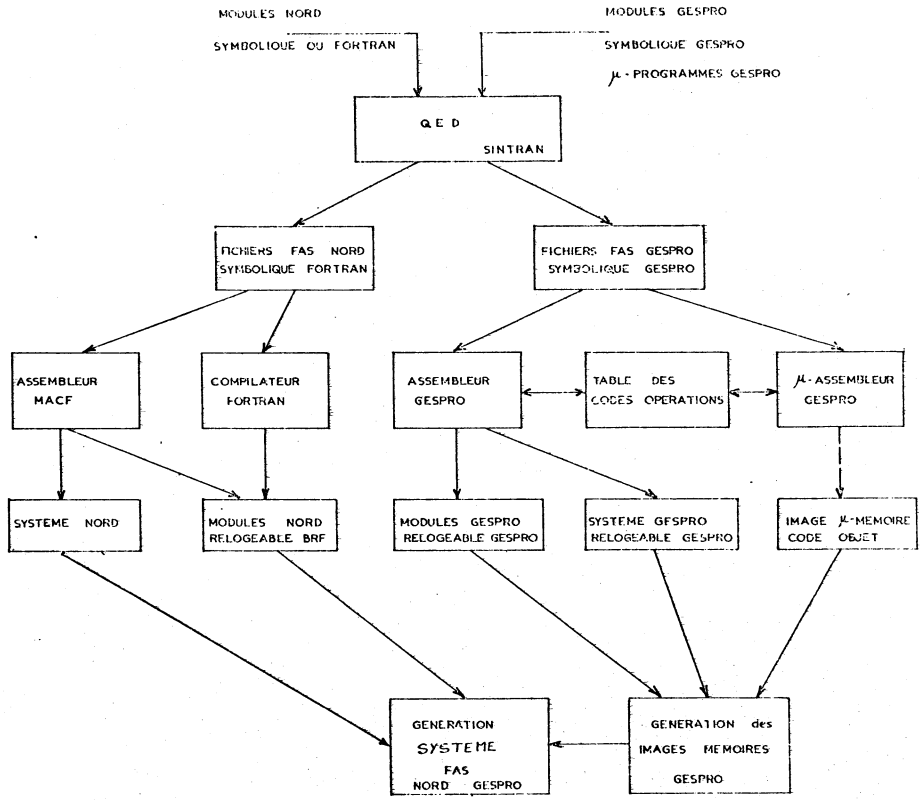


Fig.I.6

PRINCIPE DE GENERATION DU LOGICIEL FAS

en langage machine.

Les résultats sont une liste d'instructions assemblées et un programme objet.

4.4. Programmation de GESPRO

La programmation de GESPRO est tout à fait spéciale en ce sens qu'elle est le point d'interaction entre deux processeurs différents GESPRO et NORD 10, et qu'elle doit assurer un certain dialogue entre les deux machines (fig.I 7)

Elle consiste en une partie classique, formée de programmes relogeables :

- le système d'exécution spécialisé à l'acquisition, assurant la synchronisation des tâches avec l'expérience
- le programme d'acquisition composé de modules d'acquisition spécifiques à chaque type de détecteurs, et adaptable à chaque phase de l'expérience.

Une partie fixe pour permettre l'accès de NORD-10 est composée :

- d'un "buffer" événements
- d'un "buffer" contenant les paramètres système, qui définissent l'acquisition

Ils sont définis par l'opérateur en début d'expérience et chargés en mémoire GESPRO par NORD.

Il en résulte une prise en charge par le système de développement d'une partie de la gestion de la mémoire pour surmonter ce problème de coexistence de parties fixes et relogeables en mémoire.

III. CONCLUSION

La structure multiprocesseurs du système d'acquisition F.A.S. avec la microprogrammabilité du processeur : GESPRO et la nécessité du développement sur NORD-10 sont les deux caractéristiques fondamentales qui vont définir le cahier des charges relatif à notre système de développement de programmes.

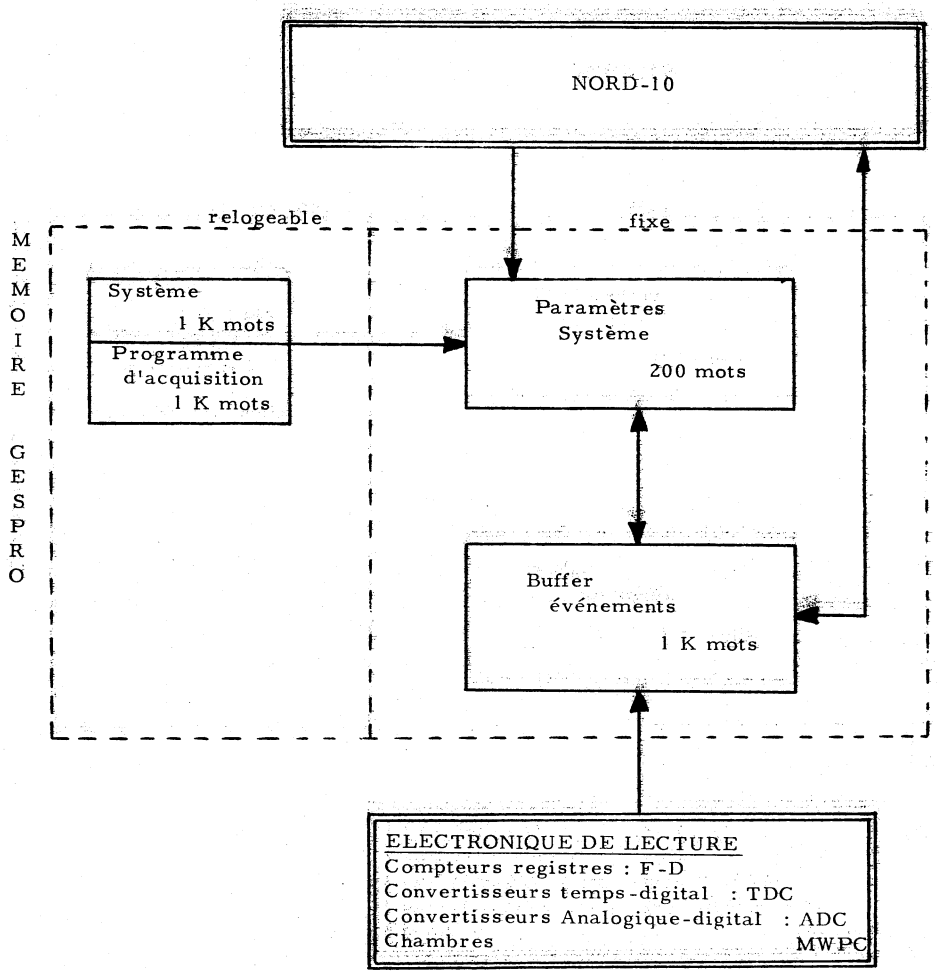


Fig.I.7 : Structuration de la mémoire GESPRO

CHAPITRE II

ELABORATION DU SYSTEME DE DEVELOPPEMENT

I. INTRODUCTION

La programmation du système F.A.S. à structure multiprocesseurs, implique le développement de programmes à chaque niveau où se trouve un processeur.

En ce qui concerne GESPRO, ne possédant pas de système d'exécution élaboré et ne possédant pas de dialogue avec les périphériques, le support principal du système de développement sera obligatoirement le mini-ordinateur NORD-10 permettant l'utilisation de l'édition de texte et de la mémoire de masse principalement.

Des considérations vues au chapitre précédent, il résulte que le système de développement devra assurer les différentes tâches suivantes :

- la gestion des jeux de définition d'instructions
- l'édition de texte
- l'assemblage à partir de jeux d'instructions adaptés à l'expérience
- l'édition de liens entre modules relogeables
- le chargement par CAMAC des modules relogés
- le contrôle mémoire de GESPRO
- l'option "debugging", avec possibilité d'exécution à partir de NORD-10

II. GENERALITES

Le langage de programmation le plus élémentaire c'est à dire le plus proche du langage machine (binaire) est le langage assembleur, et le programme qui le traduit en langage machine s'appelle assembleur. Un assembleur traduit des instructions en langage symbolique sous forme de module source en instructions en langage machine une à une. Le résultat de la traduction est un module objet.

L'objet produit, peut être directement chargeable en mémoire pour exécution à une adresse fixe ; le programme est dit alors en format binaire absolu. Mais cette méthode est très lourde et très rigide et pratiquement inutilisable pour l'exécution simultanée de plusieurs modules objet.

C'est pourquoi la presque totalité des assembleurs produisent des programmes dit en format binaire relogeable, c'est à dire chargeables en mémoire à une adresse quelconque après une édition de liens des différents modules constituants.

L'éditeur de liens, à partir d'une adresse de chargement, calcule et translate les adresses à l'intérieur de chaque module de façon à relier et à placer les uns derrière les autres les différents modules créant ainsi un module chargeable (voir fig.II-1).

L'élaboration d'un programme, de l'écriture à l'exécution passe par 3 stades fondamentaux au cours desquels ils sont stockés en mémoire de masse.

Avant assemblage au niveau langage symbolique, les programmes sont stockés sous forme de module Source. L'avantage de ce stockage est que l'on dispose de programmes facilement modifiables, mais par contre non directement chargeables.

Après assemblage au niveau binaire relogeable, les programmes sont stockés sous forme de module objet. A ce stade les programmes ne sont ni modifiables ni chargeables directement ; mais par contre après le temps d'édition de liens ils sont chargeables à une adresse quelconque fixée à l'édition de liens.

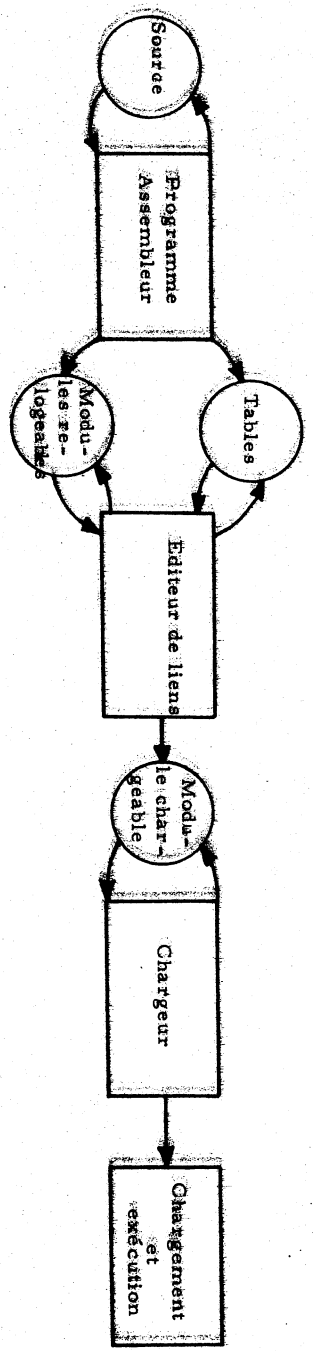


Fig.II.1 : Fonctions de l'assembleur, éditeur de liens et chargeur

Après l'édition de liens au niveau binaire absolu, les programmes sont stockés sous forme de modules image-mémoire ou modules chargeables. A ce niveau les programmes sont directement chargeables, mais pas modifiables facilement (ce sera le travail de l'option debugging du chargeur dans notre système de développement). Ce stockage est destiné à des programmes testés et exécutés toujours à la même adresse en mémoire centrale.

En résumé :

- Les programmes non testés et modifiables seront stockés sous forme de module source
- Les programmes testés mais chargeables à adresse variable seront stockés sous forme de module objet
- Les programmes testés et chargeables à adresse fixe seront stockés sous forme de module chargeable.

Les différents éléments "software" d'un système de développement au niveau système opérationnel se répartissent comme indiqué sur la figure II.2 :

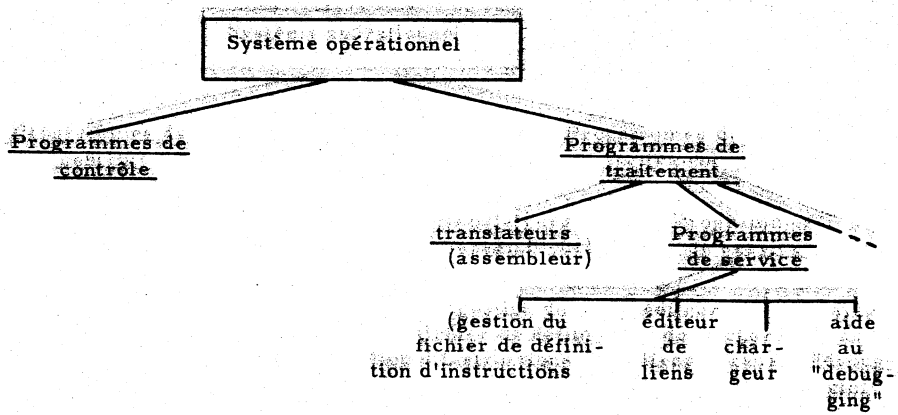


Fig.II.2. Structure logicielle d'un système de développement dans le cadre du système d'exécution d'un ordinateur

Les paragraphes suivants de ce chapitre vont être consacrés aux différents stades d'élaboration de chacun des éléments du système de développement. Ce système devra répondre exactement aux exigences de l'expérience c'est à dire du système F.A.S. et GESPRO dont nous avons vu les principales caractéristiques, et aux exigences des spécialistes systèmes de NORD-10 et GESPRO, pour lesquels ce système est destiné.

III. STRUCTURE GENERALE DU SYSTEME A ADOPTER

1. Introduction

En dehors des caractéristiques relatives à F.A.S. et GESPRO, le système de développement devra réunir les caractéristiques générales suivantes :

- fiabilité
- facilité d'utilisation (entrées, résultats, commandes équivalentes à celles de NORD-10)

avec des performances maximum du point de vue :

- vitesse d'exécution
- occupation de place en mémoire centrale et en mémoire de masse

Dans la suite de cette étude ces caractéristiques représenteront des contraintes sur chaque fonction du système définies par le cahier de charges, et nous réaliserons une optimisation sur le temps d'exécution et la place occupée en mémoire.

2. Cahier de charges :

A partir des caractéristiques de F.A.S. et GESPRO détaillées dans le premier chapitre, nous allons nous définir un cahier de charges, ou en d'autres termes, les fonctions que devra assurer le système de développement.

2.1. Cahier des charges relatif à (F.A.S.)

a) Structure multiprocesseurs

Le système pouvant s'appliquer à des processeurs différents devra posséder tous les caractères de généralités .

- l'accès à la mémoire de GESPRO par d'autres processeurs nécessite un contrôle et une gestion de la mémoire par le système
- possibilité à d'autres processeurs ou système d'exécution de charger des programmes en mémoire GESPRO, d'où une structure d'image mémoire généralisée.

b) caractère évolutif de F.A.S. :

Le logiciel de F.A.S. sera en constante évolution signifiant pour GESPRO des jeux d'instructions adaptés à la manipulation, ce qui nécessite :

- une gestion dynamique des fichiers relatifs aux instructions et pris en compte par l'assembleur
- les longueurs de programmes à traiter devront pouvoir varier de quelques mots jusqu'à la limite d'adressage 16 bits soit 64 K mots

* en temps que système de développement ce qui signifie mise au point, tests correction de programmes, il devra fournir une large gamme d'utilisation :

- assemblage en interactif
- assemblage avec listing
- assemblage sans listing
- assemblage avec listing partiel
- assemblage conditionnel
- listing ou non des tables
- listing des variables
- commentaires
- option de debugging au niveau assembleur par un jeu de diagnostics ; au niveau chargeur par la possibilité de visualiser, modifier des mots mémoire, d'exécuter les programmes.

2.2. Cahier des charges relatif à GESPRO

a) Processeur à mots de 24 bits

Sur les 24 bits, 16 étant réservés à l'adressage, ce qui donne la possibilité de faire de l'adressage absolu. Dans l'instruction c'est donc l'adresse réelle qui figurera.

Les constantes logiques occuperont 24 bits

Un mot GESPRO sera équivalent à un double mot NORD avec 8 bits ignorés

b) Instructions à plusieurs mots :

Les instructions à plusieurs mots équivalent à des macroinstructions au niveau interne à la machine. Donc pas de macro au niveau assembleur.

c) Processeur microprogrammable :

Les jeux d'instructions ne seront pas figés, et les fichiers correspondants devront pouvoir être mis à jour facilement sans pour autant être obligé de modifier quoi que ce soit au niveau de l'assembleur.

d) Pas de système d'exécution élaboré :

L'assembleur et le chargeur devront avoir une structure de recherche des erreurs très élaborée car à l'exécution aucune sécurité n'est assurée.

- au niveau assembleur, jeu de diagnostics élaboré par recherche systématique des erreurs de syntaxe et de sémantique
- au niveau chargeur, des diagnostics d'erreurs, une visualisation et un contrôle de l'implantation en mémoire.

e) Structure modulaire du logiciel :

Le système de développement devra présenter à chaque niveau de stockage une structure de fichiers source, objet, image mémoire adaptable à une structure modulaire des programmes donnant les différentes possibilités suivantes (figure II.3)

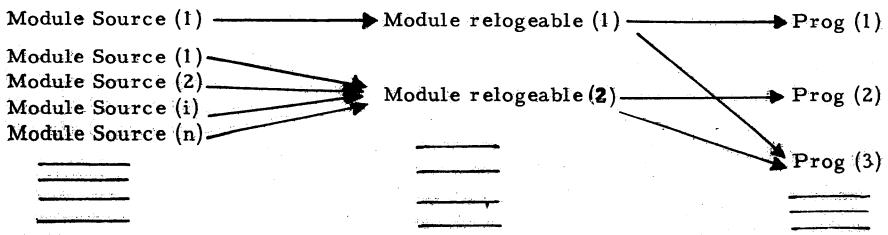


Fig. II.3

Ceci implique un fichier au niveau source ou objet pour un ou plusieurs modules et au niveau chargement, une structure reliable des différents modules.

3. Etude critique de l'assembleur existant :

3.1. Introduction

Au stade actuel d'avancement de l'expérience de physique, l'assembleur existant a suffisamment été utilisé, pour nous servir de point de départ, par une étude critique, pour le développement de notre système.

3.2. Caractéristiques principales de l'assembleur existant

C'est un cross-assembleur, écrit en FORTRAN exécuté en un seul passage sur le mini-ordinateur NORD-10 à mot de 16 bits sous le système d'exécution SINTRAN III.

Il traduit le langage symbolique GESPRO, à partir des jeux d'instructions existants, en langage binaire relogeable en mots de 24 bits.

3.3. Défauts de cet assembleur :

C'est un assembleur extrêmement lent, environ 1 seconde par instruction. Il est d'utilisation difficile, à cause du grand nombre de paramètres à donner en entrée et du manque de communiation avec l'opérateur.

La programmation est rendue difficile par des directives non judicieuses.

Les résultats de l'assemblage sont présentés sour une forme le plus souvent inutilisable par l'opérateur. De plus il n'est pas fiable à 100 % et ne permet pas d'assembler des programmes de plus de 200 instructions source en une seule fois.

3.4. Cause des défauts

Une étude détaillée de cet assembleur nous a permis de définir les points principaux qui sont la cause de l'insuffisance des performances :

- en temps d'exécution
- en capacité d'assemblage (longueur des programmes)
- en facilité d'utilisation

Ces points critiques sont les suivants :

- écriture des programmes en langage Fortran :
  ce langage a l'inconvénient d'être lent (facteur 3 à 5 par rapport à l'assembleur) et d'occuper une place mémoire importante (facteur 3 à 5 par rapport à l'assembleur)
- structure des tables :
  . surutilisation de tables dûe à leur nature, et à leur structure.
  . défaut de gestion de ces tables (trop petites et entièrement résidentes).
  . la table des symboles principalement a une structure interne non adaptée à la recherche rapide de symboles (recherche linéaire)
- organisation des fichiers :
  . l'accès aux différents fichiers source, objet, définition d'instructions, est fait à chaque décodage d'instruction. C'est la cause primordiale de perte de temps, les temps d'accès étant généralement longs (ta≈60 ms)
  . la lecture ou l'écriture sont faites en mode séquentiel, nécessitant un accès pour chaque mot lu ou écrit.
  . l'organisation du fichier objet avec une structure de sous-fichiers complique son utilisation et donc le temps de manipulation.
  . la mise à jour des fichiers de définitions d'instructions est faite en partie par des directives, qui compliquent la programmation GESPRO.
  . la gestion générale des fichiers par des directives (cree, delete, consrv,...) et leur structure interne, sont telles que les temps de recherche et de compactage sont beaucoup trop grands.
  . le manque important du caractère conversationnel dans cet assembleur, rend son utilisation difficile et son efficacité discutable.

3.5. Structure de notre système devant en résulter

A partir des éléments précédents nous allons élaborer une macro-structure du système de développement optimale du point de vue vitesse d'exécution et occupation de place mémoire

- l'utilisation au maximum du langage assembleur par rapport au Fortran doit déjà nous faire gagner un facteur 2 à 5 sur le temps d'exécution et la place mémoire occupée
- la limitation au minimum des accès sur disque, par un jeu de tables approprié et une mise en résident de ces tables donnant le meilleur compromis entre place mémoire et temps d'exécution.
- la minimisation du nombre de fichiers et tables utiles.
- la structuration interne des tables, adaptée à leur mode d'accès, pour permettre une recherche rapide à l'intérieur, indépendamment de leur capacité.
- l'utilisation au maximum des facilités de gestion de fichiers apportés par le système opérationnel de NORD-10 par
  . l'utilisation exclusive de fichiers NORD
  . l'accès aux fichiers exclusivement en mode direct . La lecture et l'écriture se faisant par blocs de mots.
  . structuration interne des fichiers adaptée à ce mode de lecture et écriture.
- par rapport à l'ancien assembleur, il y aura suppression des macro-instructions, à cause de leur non-utilisation, GESPRO étant un processeur à instructions de plusieurs mots.

  Il y aura conservation de l'assemblage en un seul passage, étant la méthode la plus rapide par rapport au double passage
- la gestion du fichier de définitions d'instructions et l'assemblage apparaissent comme deux problèmes nettement différents, d'où l'idée de les séparer en deux programmes indépendants ce qui devra avoir pour conséquences :
  . une plus grande facilité de génération des instructions
  . une simplification de la programmation de GESPRO
  . un gain de temps et de place mémoire à l'assemblage.

4. Outils à notre disposition :

GESPRO ne possèdant pas de système opérationnel élaboré, c'est le mini-ordinateur NORD-10 qui va servir de support au système de développement.

Afin de tenir compte efficacement des contraintes de fiabilité et facilités d'utilisation avec des temps d'exécution et place mémoire minimum il est primordial d'utiliser toutes les facilités apportées par le système opérationnel SINTRAN III de NORD 10.

Ces facilités principales consistent en :

- un éditeur de texte (Q.E.D.)
- un système de gestion de fichiers (File-system)
- un jeu d'instructions assembleur très performant.

4.1. Nord-10 : [4]

NORD-10 est un mini-ordinateur de NORSK DATA, très performant du point de vue logiciel, car il a un système d'exécution très élaboré. C'est un mini-ordinateur à mot de 16 bits microprogrammé

4.2. SINTRAN III [5]

C'est un système opérationnel modulaire pour les mini-ordinateurs NORD-10, adaptable à une grande gamme de configurations (du CPU, 16 K mots de mémoire, un disque et une console, à 256 K mots de mémoire sans limitation sur le nombre et le type des périphériques.

L'orientation générale de ce système est axée sur les contrôles de processus et les applications scientifiques. Il est conçu pour l'exécution en multiprogrammation de tâches en mode non prioritaire (développement de programmes, mise au point) ou en temps réel.

Différents niveaux de programmation sont acceptés :

- le FORTRAN, standard ANSI et une extension temps réel ISO
- le BASIC
- le NODAL (langage interprétatif, haut niveau, pour process control)
- le NORD-PL (langage de niveau intermédiaire)
- le langage assembleur MAC.

Il met à la disposition des utilisateurs une très grande gamme de programmes utilitaires.

4.3. Editeur de texte Q.E.D. :

C'est un programme interactif permettant de faire de l'édition de texte. Il a été développé par NORSK DATA et fait partie de SINTRAN en tant que programme de service.

La rentrée de l'information à partir d'une console est facilitée par une série de commandes et de contrôles.

C'est donc avec ce programme que sont constitués tous les modules source aussi bien pour NORD que pour GESPRO.

4.4. Facilités du point de vue programmation :

SINTRAN III offre une série de 256 "monitor call" faisant partie intégrante du système, appelable par FORTRAN | CALL NAME (PAR, ...)| ou en assembleur par (MON nn).

Ces appels moniteurs exécutent des tâches spécifiques telles que entrée ou sortie d'un caractère sur un terminal.

A chaque appel "monitor call", à la différence d'un appel de routine de service, il y a interruption au niveau 14 et débranchement à l'élément de système désigné.

La figure II.4 donne quelques temps d'exécution d'instructions :

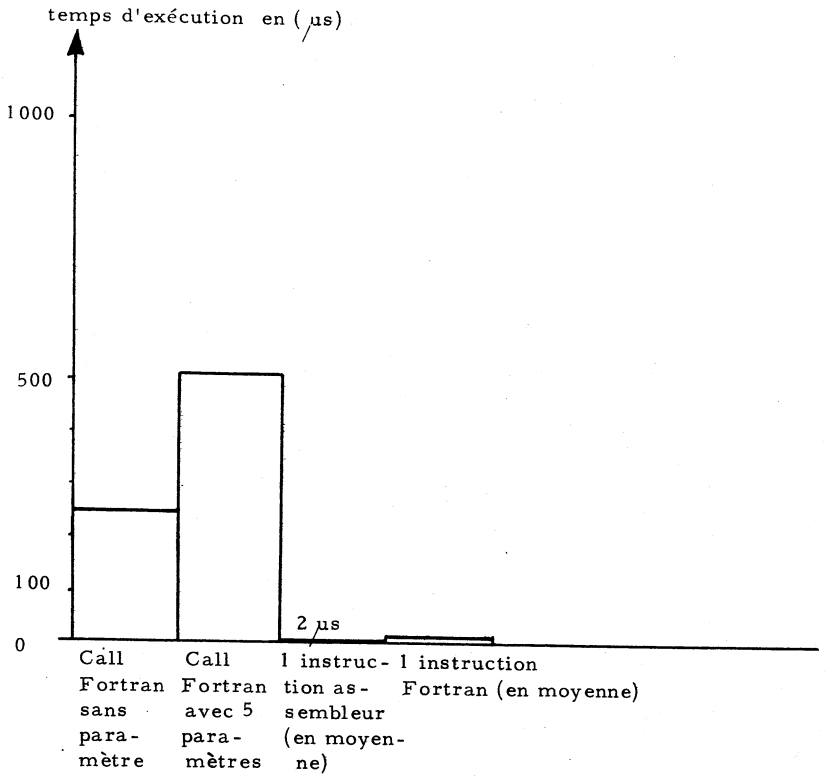


Fig. II. 4 : Principaux temps d'exécution d'instructions

- les temps d'exécution des "monitor call" s'étendent de 1 à plusieurs centaines de (ms) suivant leur nature.

4.5. Gestion de fichiers (File-system) [6]

Le système de gestion de fichiers offre à l'utilisateur de la mémoire de masse (disque, bande magnétique), une très grande facilité de manipulation de fichiers permanents, temporaires et de fichiers associés aux périphériques, avec différents degrés de sécurité.

Tous ces fichiers peuvent être accédés en séquentiel (accès de blocs de mots consécutifs) ou en "random access" (accès direct à un bloc de mots).

L'accès au "file system", se fait par une série de commandes, qui sont utilisables par programme avec un CALL ou un MON

Mesure des temps d'exécution

Afin de pouvoir faire par la suite, une optimisation en temps des différents éléments du système de développement, il nous faut connaître les temps moyens d'exécution des principales commandes.

C'est ce que nous avons mesuré ; les résultats sont représentés sur la figure II. 5.

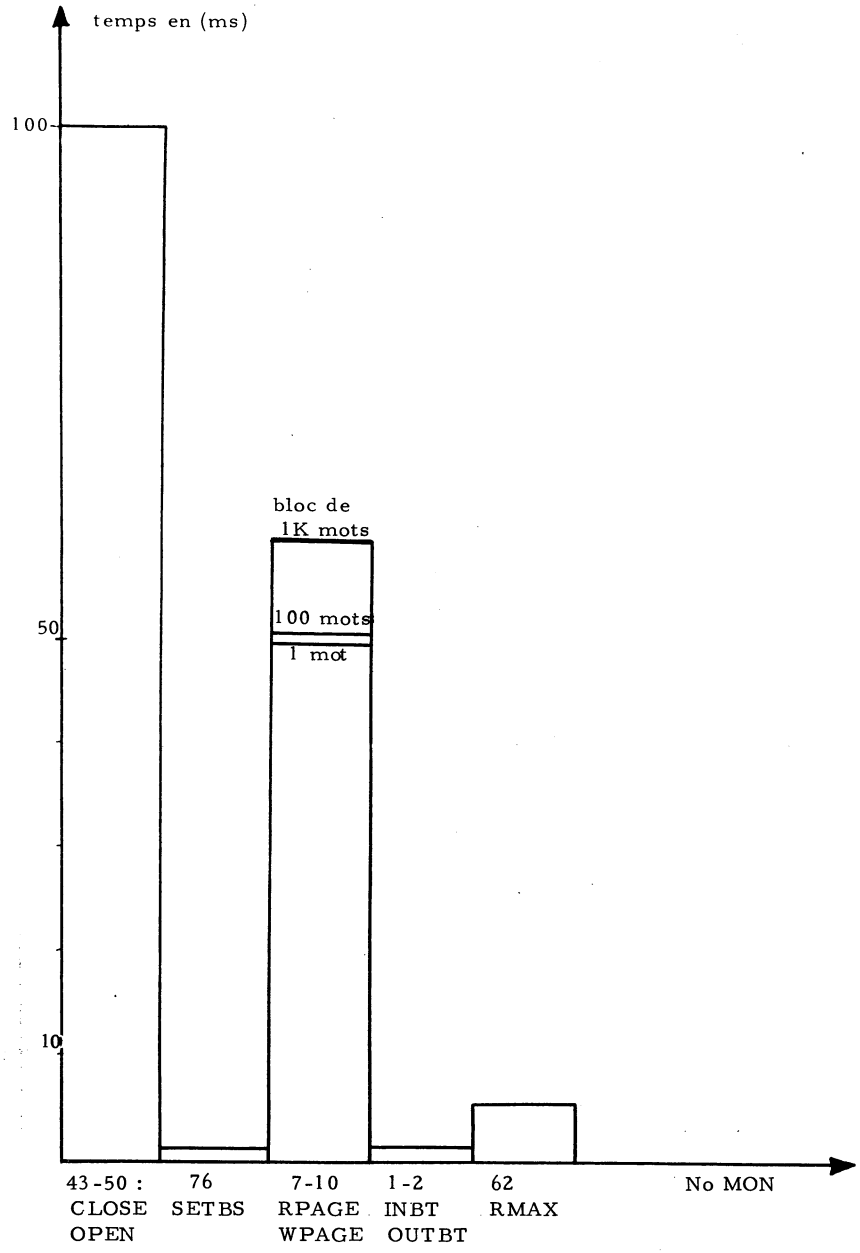


Fig. II. 5 : Temps d'exécution des principaux "monitor call"

Le temps d'exécution d'une lecture ou écriture d'un bloc de mots se compose comme suit (figure II. 6 et annexe A)

- temps d'exécution du logiciel
- temps d'accès sur disque :
  - . temps de positionnement (moyen : 35 ms)
  - . temps de latence (1/2 tour de disque en moyenne soit 12, 5 ms)
- temps de transfert (8, 2 µs/mot)
  - . 8, 5 ms pour un bloc de 1 K mots

La lecture ou écriture en accès séquentiel par rapport à l'accès direct demande un temps supplémentaire par mot de l'ordre de 11 ms.

En fonction du nombre de mots transférés, le temps d'exécution en mode direct est égal à :

$$\overline{t_n} \simeq 60 + 0,01\ n \text{ (ms)}$$

en mode séquentiel il est égal à :

$$\overline{t_n} \simeq 60 + 10\ n \quad \text{(ms)}$$

L'influence du nombre de mots transférés sur les temps d'exécution sont dans un rapport : 1000 entre les 2 méthodes.

Après ce passage en revue des outils utilisables nous allons étudier un autre point important, les méthodes de recherche de symboles à l'intérieur d'une table.

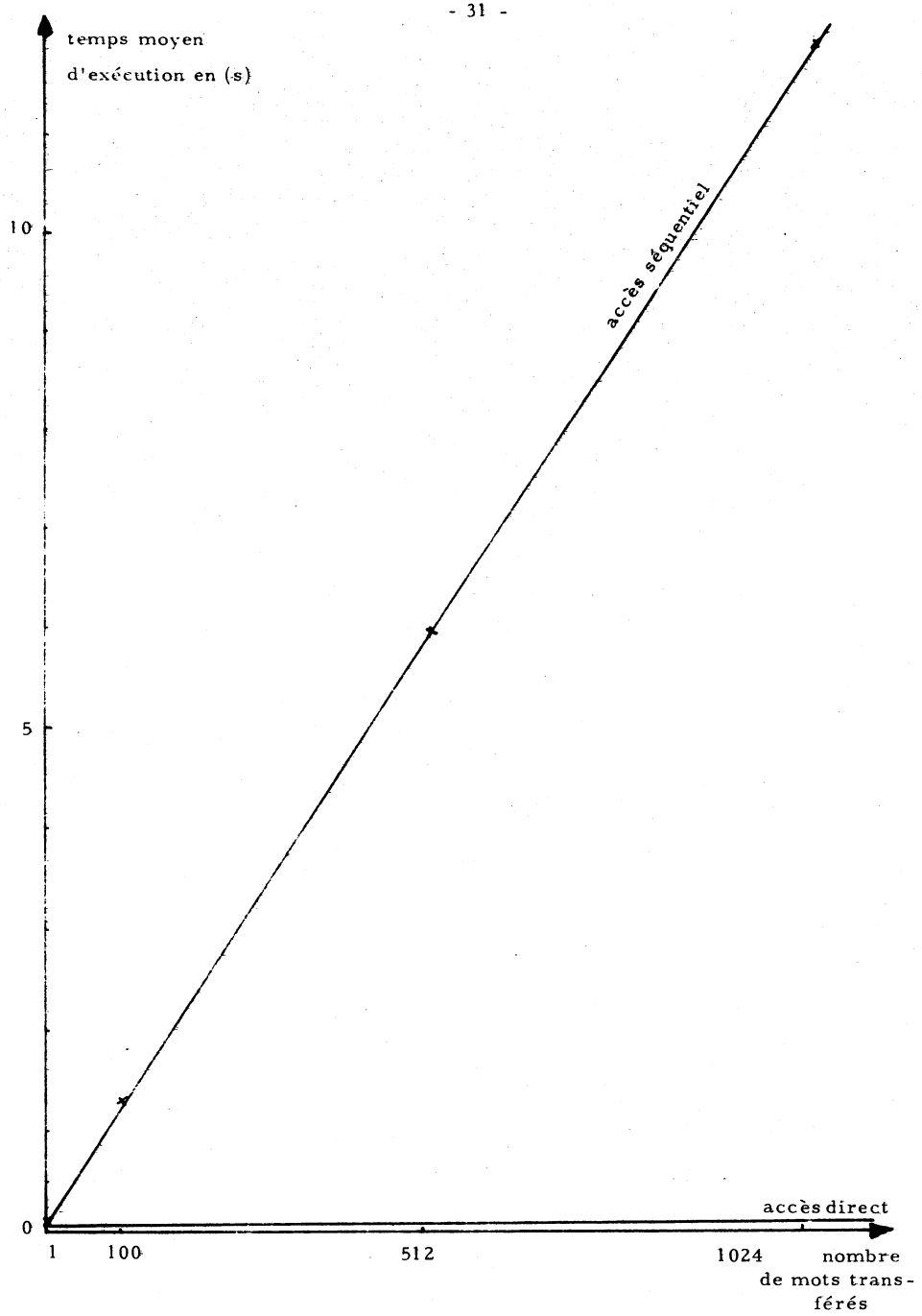


Fig. II.6 : Comparaison entre temps d'exécution en mode direct et séquentiel

5. Méthodes de recherche à l'intérieur d'une table de symboles[7] :

5.1. Introduction :

Le problème de recherche de symbôles à l'intérieur d'une table est très important, car c'est ce temps de recherche qui va pratiquement fixer le temps d'exécution des programmes.

Pour chaque élément du système de développement, le problème va se poser différemment, mais nous pouvons dès maintenant faire une étude théorique générale sur les différentes méthodes de recherche et leur utilisation.

5.2. Méthode de recherche linéaire :

Cette méthode est la plus simple et la plus facile à mettre en oeuvre. La recherche d'un symbole dans une table consiste en la comparaison successive des symboles avec celui recherché jusqu'à ce qu'on le trouve. S'il n'est pas dans la table il est ajouté à la suite des autres.

Le temps d'exécution d'une telle recherche est donc proportionnel au nombre de symboles : N, contenus dans la table, dans l'hypothèse d'une recherche équifréquentielle sur chaque symbole.

$$\overline{t_N} = k \frac{N}{2}$$

k : facteur de temps d'examination linéaire

On conçoit facilement que pour un nombre important de symboles, supérieur à 100, cette méthode devient lente. D'où la nécessité de trouver des méthodes plus élaborées permettant de gagner du temps.

5.3. Méthode de recherche logarithmique

Cette méthode consiste à ranger les symboles dans la table par ordre alphabétique, afin de les retrouver plus facilement.

C'est l'équivalent de la méthode du dictionnaire. Une recherche visuelle dans ce cas est extrêmement rapide quel que soit le nombre de symboles.

En recherche logique nous allons procéder de la même manière qu'en recherche visuelle c'est à dire par élimination de zônes successives (recherche par dichotomie).

Le temps de recherche sera donc le temps de convergence vers 1 de la suite : $\frac{N}{2^n}$

$$\frac{N}{2^n} \longrightarrow 1$$

Soit $n \simeq \frac{\log N}{\log 2}$ $\quad \bar{n} \simeq \frac{\log N/2}{\log 2} = \frac{\log N}{\log 2} - 1$

donc $\boxed{\bar{t}_N \simeq k'\left(\frac{\log N}{\log 2} - 1\right)}$ pour $N > 1$ $\quad k'$: facteur de temps d'examination logarithmique

Ce temps est proportionnel au logarithme du temps de recherche linéaire. Cette méthode est appelée Méthode de recherche logarithmique ou binaire.

Elle est très peu sensible au nombre de symbôles, mais son inconvénient majeur, est la difficulté de rangement des symbôles par ordre alphabétique à cause de l'insertion de symbole devant un grand nombre d'autres symbôles, nécessitant la translation de ceux-ci.

C'est pourquoi nous allons comparer à cette méthode les performances d'autres méthodes intermédiaires entre recherche linéaire et recherche logarithmique.

5. 4. Méthodes intermédiaires

Ces méthodes consistent principalement à diviser la table en éléments permettant la sélection de zônes précises pour la recherche proprement dite et l'insertion des symbôles.

L'orientation vers une zône particulière peut être effectuée de différentes manières en fonction de la syntaxe des symboles.

5. 4. 1. Méthode semi-logarithmique

Une méthode consiste par exemple à diviser la table en 26 parties, chacune correspondant à une lettre de l'alphabet, constituant le premier caractère des symboles (fig. II. 7).

La recherche dans ce cas peut se faire en linéaire pour ce qui est de la sélection d'une zône, et en logarithmique pour la recherche à l'intérieur

des zônes.

Si les zônes ont une capacité fixe, l'insertion ne demande la translation que d'un nombre limité de symbôles.

Exemple :

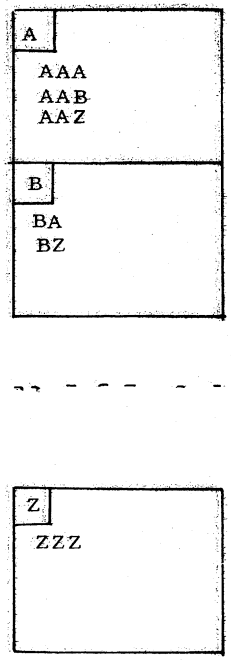


Fig. II. 7

temps d'exécution moyen :

$$\boxed{\bar{t}_N = k'' \frac{26}{2} + k'\left[\frac{\log N/26}{\log 2} - 1\right]}$$

pour $N > 26$ $\quad k''$: facteur de temps comparaison d'un caractère

L'inconvénient de cette méthode est le risque de perte de place importante, à cause de la discontinuité de rangement des symbôles.

Une autre méthode générale voisine de la méthode semi-logarithmiques, dite "hashing method".

5. 4. 2. "Hashing method" [8]

Cette méthode est basée sur la division en zônes de la table comme précédemment. La sélection d'une zône est assurée par une fonction de sélection associant des coefficients à un groupe de symboles de syntaxe équivalente.

Exemple précédent :

A ⟶ 1

B ⟶ 2

—

Z ⟶ 26

Remarque :

Le nombre total de coefficients : n, ne doit pas être petit devant le nombre total de symbôles.

La recherche et insertion de symbôle est faite de la façon suivante :

a) calcul de la fonction : $n \overleftarrow{\text{coefficient}} = M(K) \overleftarrow{\text{symbole}}$

b) orientation à la position : n de la table

Si cette case est vide, ou contient le symbole recherché, on place le symbole ou on récupère le contenu de la case et c'est terminé.

c) Sinon on fait : n = n + p (mod N) et on revient en : b

Le temps de recherche ne sera pas trop grand pour un facteur de remplissage de la table inférieur au 2/3 de sa capacité (fig. II. 8).

Diagramme :

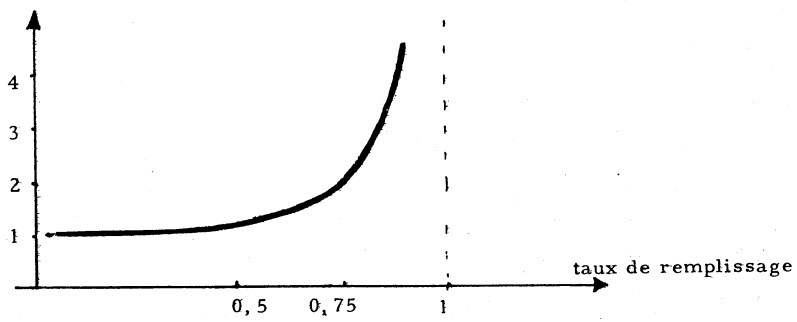


Fig. II. 8

Temps de recherche moyen :

En supposant que la fonction de sélection travaille par comparaison de caractères alphabétiques, et si n est le nombre de coefficients adoptés, on aura :

$$\frac{n}{26^{n'}} \longrightarrow 1$$

$$n' = \frac{\log n}{\log 26}$$

Le nombre de comparaisons moyen sera : $\frac{26}{2}\ n'$

$$\overline{t_N} \# \left(\underbrace{\frac{N}{n}}_{\text{approximation pour un facteur de remplissage inférieur à 2/3}} + \frac{26}{2}\ k''\right) \log_{26} n$$

k" : facteur de temps de comparaison d'un caractère

Pour $n = 26^{n_o}$

$$t_N \# \left(\frac{N}{n} + \frac{26}{2}\ k''\right) n_o$$

- application à l'exemple vu pour la méthode semi-logarithmique :

ordre d'insertion

(1)
(2)
(3)

A	AB
B	B
C	AA
D	
E	EE
Z	

$$\overline{t_N} \# \frac{N}{26} + 13\ k''$$

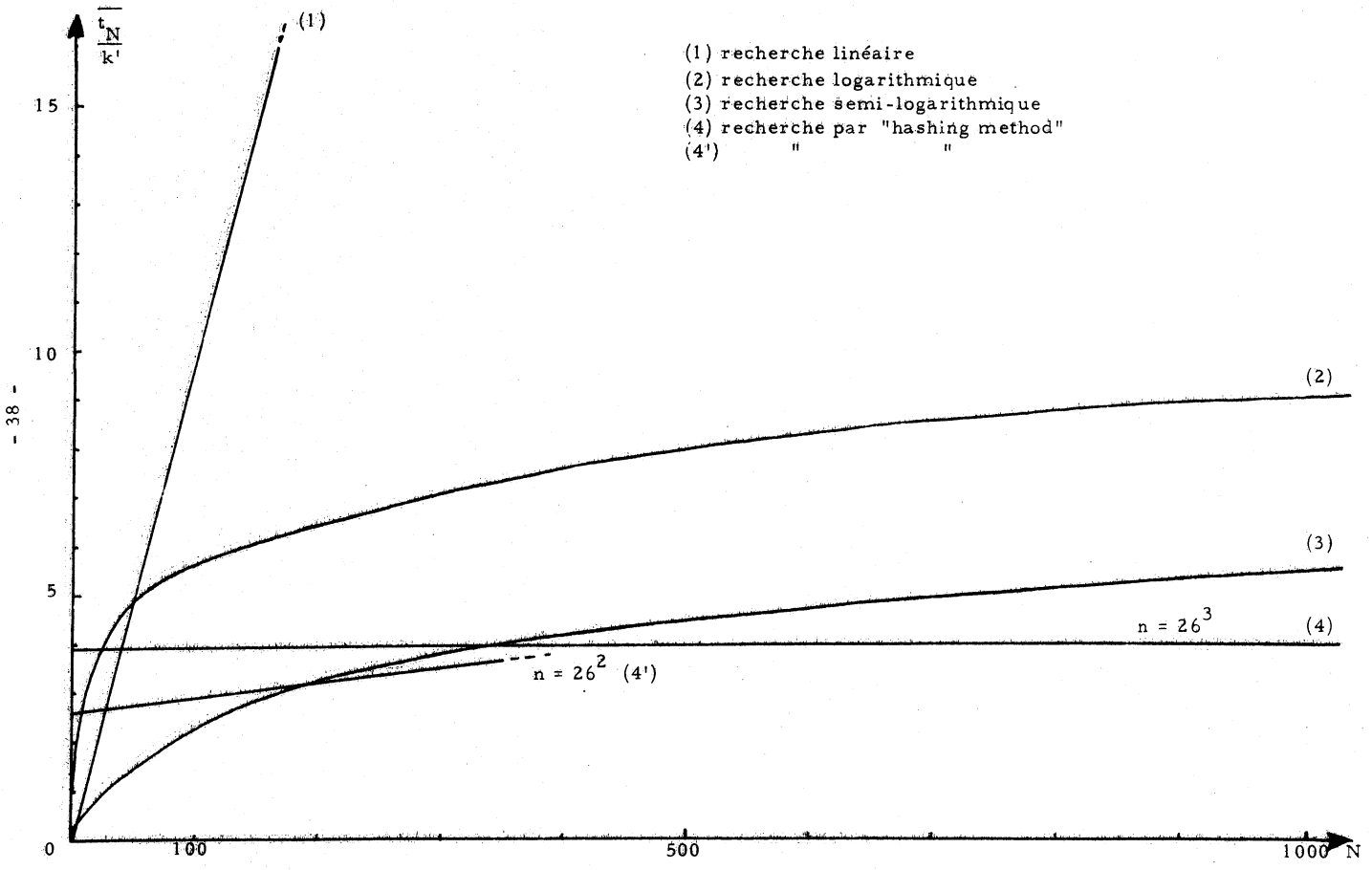


Fig. II. 9 : Comparaison de temps de recherche dans une table de symboles par différentes méthodes

5. 5. Comparaison des différentes méthodes (fig. II. 9)

$k' = 5\,k$: rapport des temps d'examination moyens entre méthode de recherche logarithmique et méthode de recherche linéaire

$k'' = \frac{k}{2}$: rapport des temps de comparaison moyens entre un caractère et un mot de 6 caractères maximum.

Les courbes de la figure II. 9 montrent que :

- la recherche linéaire est performante en temps pour un nombre de symboles inférieur à 30
- la recherche par "hashing method" est la plus rapide (facteur 2 à 2, 5 avec la recherche logarithmique), mais à la condition d'un facteur de remplissage n'excédant pas 2/3.
- la recherche semi-logarithmique est aussi plus rapide, d'un facteur 1, 5 à 2 avec la recherche logarithmique.

5. 6. Conclusion

Pour le temps de recherche toutes les méthodes exceptée la méthode linéaire, sont compétitives.

Pour le temps d'insertion de symbole la "hashing method" est la plus performante, n'effectuant pas implicitement de rangement par ordre alphabétique, donc ne nécessitant pas la translation d'information lors d'une insertion de symbole.

La méthode semi-logarithmique est plus performante en temps que la méthode logarithmique mais elle est de mise en oeuvre plus délicate et surtout se caractérise par une occupation de place mémoire importante (facteur 2 au moins avec la méthode logarithmique) et ne sera donc pas retenue.

Seules donc la méthode logarithmique et la "hashing" méthode sont retenues et le choix entre les deux se fera par la suite en fonction des applications particulières.

A partir du cahier des charges, sous les contraintes de temps d'exécution et d'occupation mémoire et en fonction des outils informatiques à notre disposition, nous allons définir une structure générale idéale du système de développement à adopter.

6. Structure générale du système à adopter : (fig. II-10)

6.1. Gestion du fichier de définitions d'instructions :

La génération des instructions, par l'introduction des définitions d'instructions dans un fichier, est en fait équivalent à la formation d'un fichier source. D'où l'idée de donner une structure au programme de gestion, voisine de celle de l'éditeur de texte Q.E.D. de NORSK DATA.

La vitesse d'exécution d'un tel programme est régie par celle d'intervention de l'opérateur, le programme principal peut donc être écrit en langage Fortran. Les parties dans lesquelles l'opérateur n'intervient pas, feront l'objet de sous-programmes écrits en assembleur.

Le rangement des mnémoniques doit être tel qu'il facilite la recherche d'une instruction au niveau utilisateur aussi bien qu'au niveau programme de gestion. Un rangement par ordre alphabétique semble être la meilleure solution.

Le fichier devra avoir une structure telle qu'il occupera le moins de place possible en mémoire, pour le mettre en résident lors de l'assemblage. Le contenu du fichier, devra être protégé contre toute fausse manoeuvre de l'opérateur et un contrôle rigoureux doit être fait sur la structure de chaque instruction rentrée, aucune erreur de syntaxe n'étant permise.

6.2. Assembleur :

Exécuté sur NORD-10 et destiné à GESPRO c'est un cross-assembleur.

C'est le programme pour lequel la vitesse d'exécution est la plus critique puisqu'au cours de son exécution, il n'y a pas intervention de l'opérateur.

Ce qui va se caractériser par :

- l'écriture du programme en assembleur
- l'assemblage en un seul passage

- l'utilisation de fichier NORD :
  - . source
  - . objet
  - . temporaire ("scratch File")
- le fichier de définition d'instructions sera mis en résident mémoire dans une table.
- la recherche dans la table de définition d'instructions sera une recherche logarithmique à partir du rangement de symboles par ordre alphabétique.
- toutes les tables devront avoir une structure telle qu'il y ait optimisation en temps d'exécution et place mémoire.
- un contrôle d'erreurs avec impression de diagnostics sera fait au cours de chaque phase d'assemblage.
- un jeu de directives et pseudo-instructions élaboré doit assurer les différentes options possibles au sein de l'assemblage.

6.3. Chargeur relogeable :

Afin de faciliter l'utilisation d'ordres CAMAC, l'écriture du programme principal en langage Fortran, avec des subroutines assembleur, serait avantageuse.

Il devra être adapté à l'expérience de physique et aura une structure résultant des caractéristiques de l'assembleur. Il sera composé des éléments principaux suivants :

- une sélection des fichiers relogeables
- un éditeur de liens
- des tests du status de GESPRO
- le chargeur proprement dit
- une map mémoire
- à chaque stade du programme un contrôle d'erreurs avec diagnostics
- l'option "debugging" avec exécution du programme

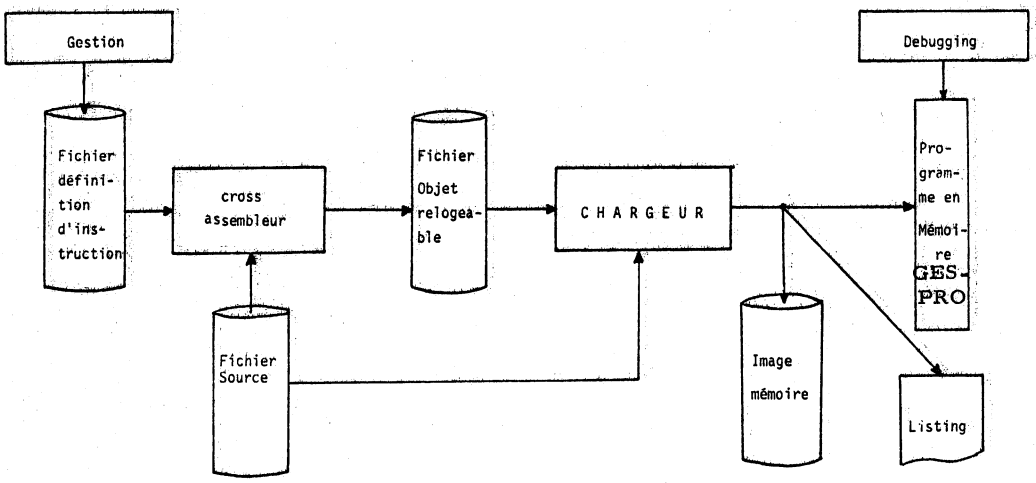


Fig. II. 10 : Structure générale du système de développement

IV. GESTION DU FICHIER DE DEFINITION D'INSTRUCTIONS

1. Introduction

L'assembleur produit un programme en format binaire en fonction de représentations symboliques d'adresses et de codes opérations.

A partir du mnémonique d'instruction, il doit donc connaître la syntaxe et en partie la sémantique correspondante.

Dans ce but on doit lui fournir un jeu d'instructions dont la définition de chacune d'elle se trouvera sous forme codée accessible et compréhensible par lui, c'est ce qu'on va appeler le fichier de définition d'instructions.

2. But :

La fonction principale du programme de gestion du fichier d'instructions est de générer les instructions sous forme de définitions dans un fichier.

Il doit donc posséder toutes les caractéristiques principales d'un éditeur de texte :

- manipulations générales sur un fichier :
  - . création physique
  - . effacement physique
  - . impression du contenu sur un terminal au choix
  - . copie du fichier dans un autre fichier
- manipulations générales sur le contenu du fichier :
  - . adjonction de ligne
  - . effacement de ligne
  - . impression de ligne
  - . modification de ligne

mais il doit aussi effectuer du traitement de caractères et d'expressions.

3. Principe

Dans le but de faciliter le traitement des données provenant de l'opérateur, celles ci vont être enregistrées en format ASCII, par un sous-

programme assurant les possibilités de correction principales, lors de la rentrée de ces données à partir de la console de visualisation.

Les corrections peuvent être effectuées grâce aux contrôles suivants :

CTRL	Signification
A	efface le dernier caractère qui vient d'être tapé
Q	efface la ligne qui est en train d'être tapée
C	recopie 1 caractère (à l'emplacement correspondant) de l'ancienne ligne
D	recopie, à partir de l'endroit considéré, l'ancienne ligne
E	permet d'insérer des caractères
S	supprime 1 caractère à l'endroit correspondant
0	recopie l'ancienne ligne jusqu'à la rencontre du caractère tapé après CTRL : 0.
L	permet de sortir de l'opération en cours

Après la prise en compte de l'expression rentrée, si c'est une commande elle est exécutée, si c'est une donnée elle est traitée conformément à sa nature, et écrite dans le fichier sur disque en format binaire. Les principales opérations que doit effectuer le programme en fonction de la commande considérée, se définissent comme suit :

- création du fichier, il crée une entête au fichier dont nous détaillerons les différents éléments par la suite.
- impression du contenu du fichier, il y a deux options de "listing" :
  - . un listing représentatif du contenu réel du fichier
  - . un listing de l'information réarrangée, facilement compréhensible par l'opérateur (ANNEXE B)

- édition lors des sous-commandes :
  - append : le programme doit assurer une protection très stricte contre toute fausse manoeuvre de l'opérateur.
    De plus toute instruction doit être acceptée que si elle est syntaxiquement correcte
  - list : le programme doit imprimer une instruction ou groupe d'instructions dans le format de rentrée à la console.
  - delete : il doit y avoir effacement de l'instruction indiquée, après contrôle, et compactage de l'enregistrement dans le fichier.
  - edit : il doit y avoir listing de l'instruction désignée et remplacement par l'instruction rentrée.

4. Structure à adopter :

4.1. Format général d'une instruction :

Les instructions GESPRO sont formées d'un ou plusieurs mots de 24 bits. Le premier mot est formé :

- d'un champ code opération sur 8 bits
- d'un champ adresse sur 16 bits

Les autres mots peuvent être formés de :

- données
- constantes
- adresses

sur un nombre quelconque de bits.

Pour la définition d'instructions nous avons structuré l'instruction en deux zônes (figure II-11)

- la zône commande, constituée par les 8 premiers bits.
- la zône argument, constituée par le reste de l'instruction (notons qu'il peut y avoir des codes opération dans les mots autres que le 1er mot d'une instruction)

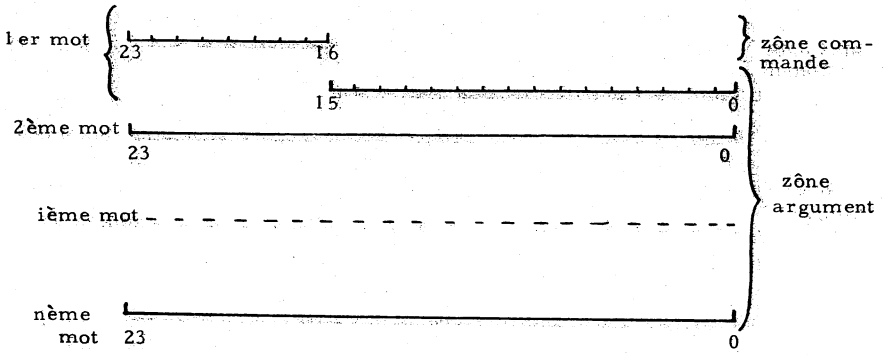


Fig.II.11 : Format général d'une instruction

4.1.1. Structure de la zône commande

La zône commande est formée d'une constante logique (le code opération) suivie pour la plupart des instructions d'une zône complémentaire représentative des registres utilisés (fig.II.12).

Cette zône est constituée d'un ou deux champs à un ou deux bits. La figure II.13 donne toutes les configurations existantes.

4.1.2. Structure de la zône argument :

Toute zône argument sera :

Type A	Constante 1 à 24 bits	Décimale octale	non signée
	Adresse sur 16 ou 24 bits		
Type D	Donnée 24 bits	Décimale octale	signée
Type E	Donnée 24 bits	Décimale octale	non signée

DESIGNATION	CODE	NATURE
A B	00 01	registres accumulateur
D E	10 11	registres index
ID	0	
IE	1	

Fig.II.12 : registres de GESPRO

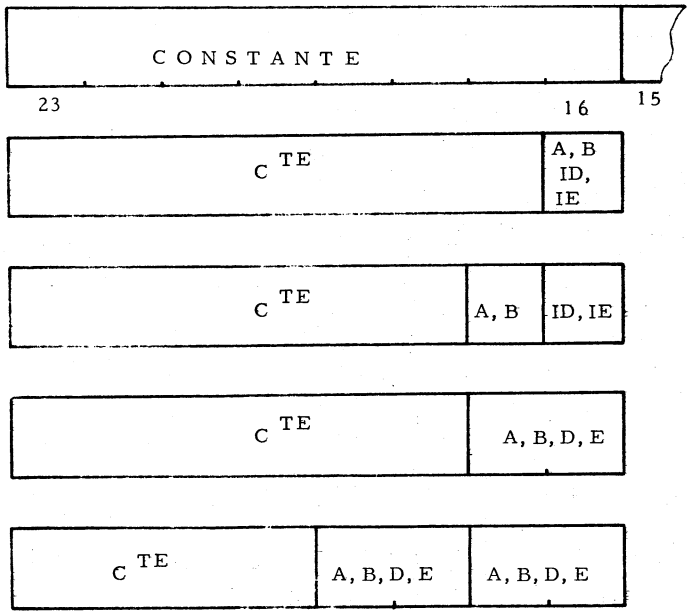


Fig.II.13 : Différentes configurations de la zône commande d'une instruction

4.2. Codage

Une définition d'instruction sert à définir les différents champs de l'instruction :

- du point de vue syntaxique : - zône occupée dans l'instruction
  - numéro dans la zône
  - longueur
- du point de vue sémantique : - nature

Dans le fichier de définitions d'instructions, chaque élément d'instruction est caractérisé par :

- un code (zône occupée et nature) figure II.14
- un indice (n° ordre dans la zône) ou la valeur pour constante
- le nombre de bits occupés par l'élément

CODE	DESIGNATION	Définition
1	élément en zône commande ou argument	Constante octale ou décimale
14	C : élément en zône commande	Donnée octale non signée
15	A : élément en zône argument	Adresse décimale ou octale non signée
16	D : élément en zône argument	donnée décimale ou octale signée
17	E : élément en zône argument	donnée décimale ou octale non signée

Fig.II.14 : Codes de définition d'instruction

Exemple :

Instruction : LD (load)

1	Code opération	6	14	1	2	15	1	16

Remarque :

L'indice indiquant le numéro d'ordre dans la zône, est l'ordre dans lequel les éléments seront générés dans le programme objet.

Il peut cependant être différent de l'ordre d'apparition des éléments dans les instructions en langage symbolique. Cela peut permettre de modifier des instructions machine tout en conservant des programmes déjà écrits ou d'écrire des instructions en langage symbolique de façon plus agréable pour l'utilisateur.

4.3. Sous-programmes de manipulation de caractères :

Le traitement des données est assuré par un jeu de sous programmes assembleur assurant : les conversions suivantes :

ASCII (décimal) ⟶ binaire
ASCII (octal) ⟶ binaire
binaire ⟶ ASCII (décimal)
binaire ⟶ ASCII (octal)
ASCII (format A2) ⟶ ASCII (format A1)
(2 bytes par mot) (1 byte par mot)
ASCII (format A1) ⟶ ASCII (format A2)

4.4. Structure générale du fichier de définitions d'instructions

Le fichier doit être structuré de telle façon que l'occupation de place soit optimale.

Pour éviter à l'opérateur de fixer une longueur arbitraire au fichier à sa création, le fichier NORD utilisé sera un fichier indexé (l'information sur le disque ne se trouvera pas en une suite continue d'enregistrements).

Cette structure permet d'augmenter la longueur du fichier au fur et à mesure de son remplissage. C'est aussi un fichier NORD de type DATA, l'information se trouvant en format binaire.

Toute suppression d'information dans le fichier fait l'objet d'un compactage.

4.4.1. Partitionnement du fichier (fig.II.15)

Un fichier en général comporte 3 catégories de zônes :

- une zône identité (nom, capacité)
- une zône pointeurs (accès à chaque zône et état)
- les zônes enregistrements proprement dites.

Dans notre cas les zônes enregistrements sont au nombre de 2. Pour faciliter la recherche des mnémoniques d'instruction, ils font l'objet d'une zône à eux seuls : (l'index). A chaque mnémonique doit être attaché une adresse permettant de retrouver la définition associée.

Les définitions sont donc stockées dans une deuxième zône : la zône renseignements ou enregistrement . A un élément d'index de 4 mots, correspond en moyenne un élément de la zône enregistrement de 13 mots. Les deux zônes seront donc dimensionnées dans le rapport 1/4.

4.4.2. Structure de l'index :

L'index constitue une zône pointeur pour la zône enregistrement proprement dite.

Cette zône maitresse du fichier fera donc l'objet de recherches pour l'accès à toute instruction par programme et du point de vue visuel sur les listings et sur la console de visualisation à l'édition de texte.

La nécessité de faciliter la tâche à l'opérateur dans la recherche d'instructions et avoir de bonnes performances en temps lors de la recherche au niveau de ce programme et surtout au niveau assembleur nous avons choisi un rangement des mnémoniques par ordre alphabétique dans l'index.

Cette solution impose une méthode de recherche logarithmique. Cette méthode reste très acceptable par rapport à une "hashing" méthode, le nombre de symboles maximum ne dépassant pas 200 à 300.

Chaque ligne d'index sera constituée de 4 mots :

- le mnémonique en format A2 sur 3 mots

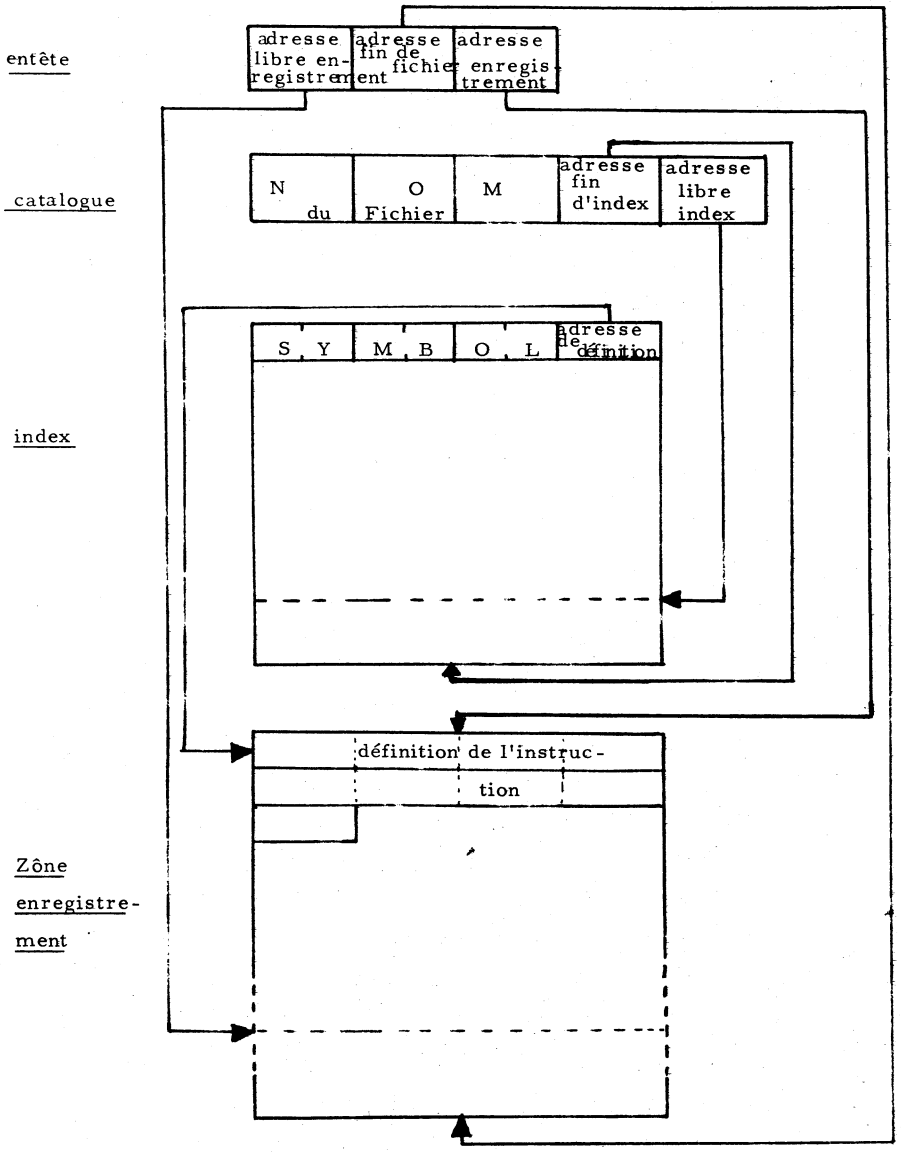


Fig.II.15 : Structure du fichier de définitions d'instructions

- l'adresse de définition dans le 4ème mot

4.4.3. Structure de la zône enregistrement :

Dans cette zône les définitions d'instructions sous forme codée sont placées les unes à la suite des autres. Chaque définition étant précédée du nombre de mots occupé par la définition.

5. Gestion du fichier

Toute manipulation sur le contenu du fichier se fera directement sur le fichier lui-même et non en résident mémoire NORD.

Cette méthode apparemment est plus longue que celle effectuant un traitement en mémoire. Mais d'une part, ce temps de traitement sera masqué entre deux interventions de l'opérateur, d'autre part une économie importante de place mémoire peut être réalisée.

Rangement et effacement d'instructions dans le fichier :

Comme nous l'avons vu précédemment c'est l'index qui sert de pointeur pour les opérations de rangement ou effacement d'instructions dans le fichier

5.1. Adjonction d'une instruction (fig.II.16)

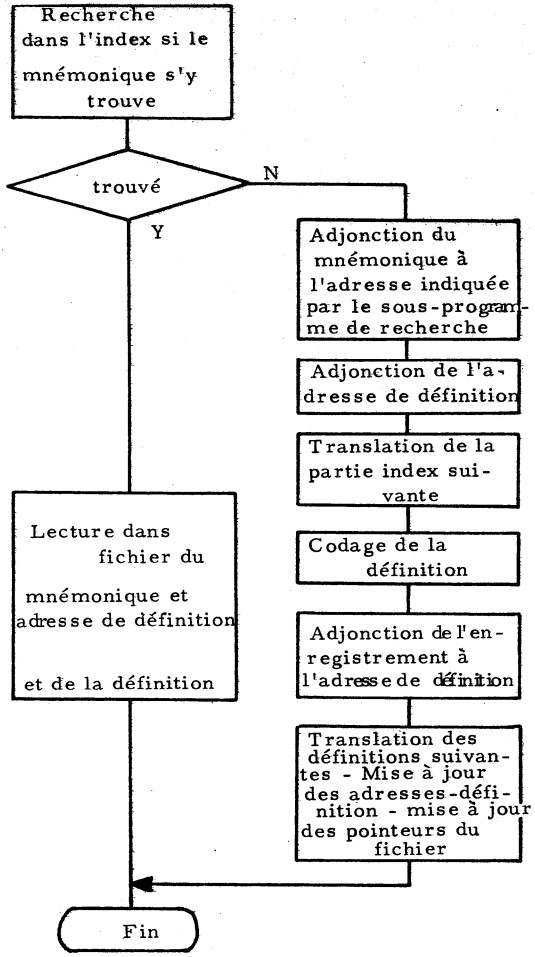


Fig.II.16 : Organigramme d'adjonction d'une instruction au fichier de définition d'instruction

5.2. Effacement d'une instruction (fig.II.17)

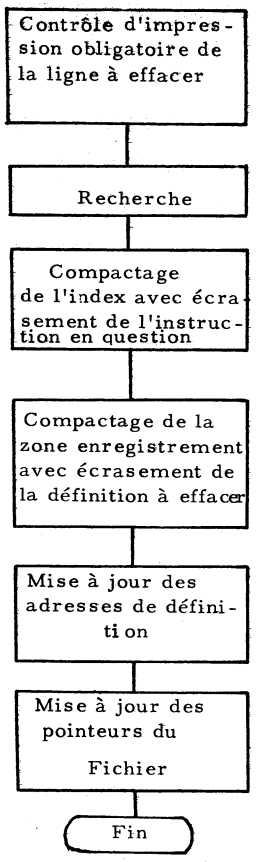


Fig.II.17 : Organigramme d'effacement d'une instruction du fichier de définition d'instruction

5.3. Sous-programme de recherche dans l'index du fichier :

Ce sous-programme est écrit en assembleur, pour une question de puissance de programmation principalement et de vitesse d'exécution.

a) Mode opératoire :

Ce programme ne fait que rechercher le mnémonique en question et transmet au programme appelant, un flag de présence s'il l'a trouvé, ainsi que l'indice dans l'index. S'il ne se trouve pas dans l'index, il transmet l'indice auquel le mnémonique doit être mis, ainsi que l'adresse à laquelle la définition de l'instruction doit être mise.

Le symbôle à rechercher se trouve en format A2 (code ASCII) sur 3 mots NORD de 16 bits ou 6 bytes. La comparaison est faite mot par mot, entre les deux symbôles.

La lecture des mnémoniques dans l'index est faite à chaque nouveau symbôle à comparer avec un RFILE de temps d'exécution moyen de 50 ms pour un bloc de 4 mots (symbôle + adresse).

b) Temps de recherche moyen parmi N symbôles : (fig.II.18)

Nombre de comparaisons moyen : $\frac{\log N}{0,3} - 1$

temps constant d'exécution d'éléments de programme :

70 instructions x 2 µs 0,14 ms

temps d'exécution d'une boucle de recherche :

30 instructions x 2 µs 0,06 ms

temps d'exécution de MON 7 à chaque boucle : 50 ms

$k = 50 \Rightarrow \overline{t_N} \# 50\left(\frac{\log N}{0,3} - 1 + 1\right)$ ↳ lié à la structure du programme

$$\boxed{\overline{t_N} \# 167 \log N} \quad \text{pour } N > 1$$

$\overline{t_N} \# 0,2$ ms pour $N = 0$

$t_N \# 50$ ms pour $N = 1$

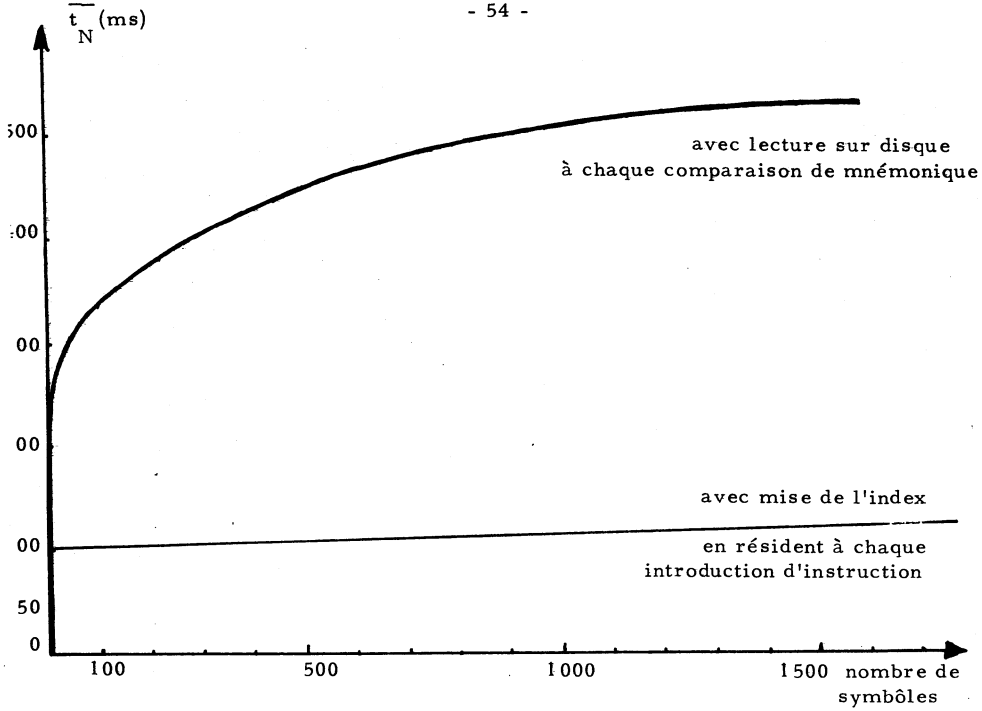
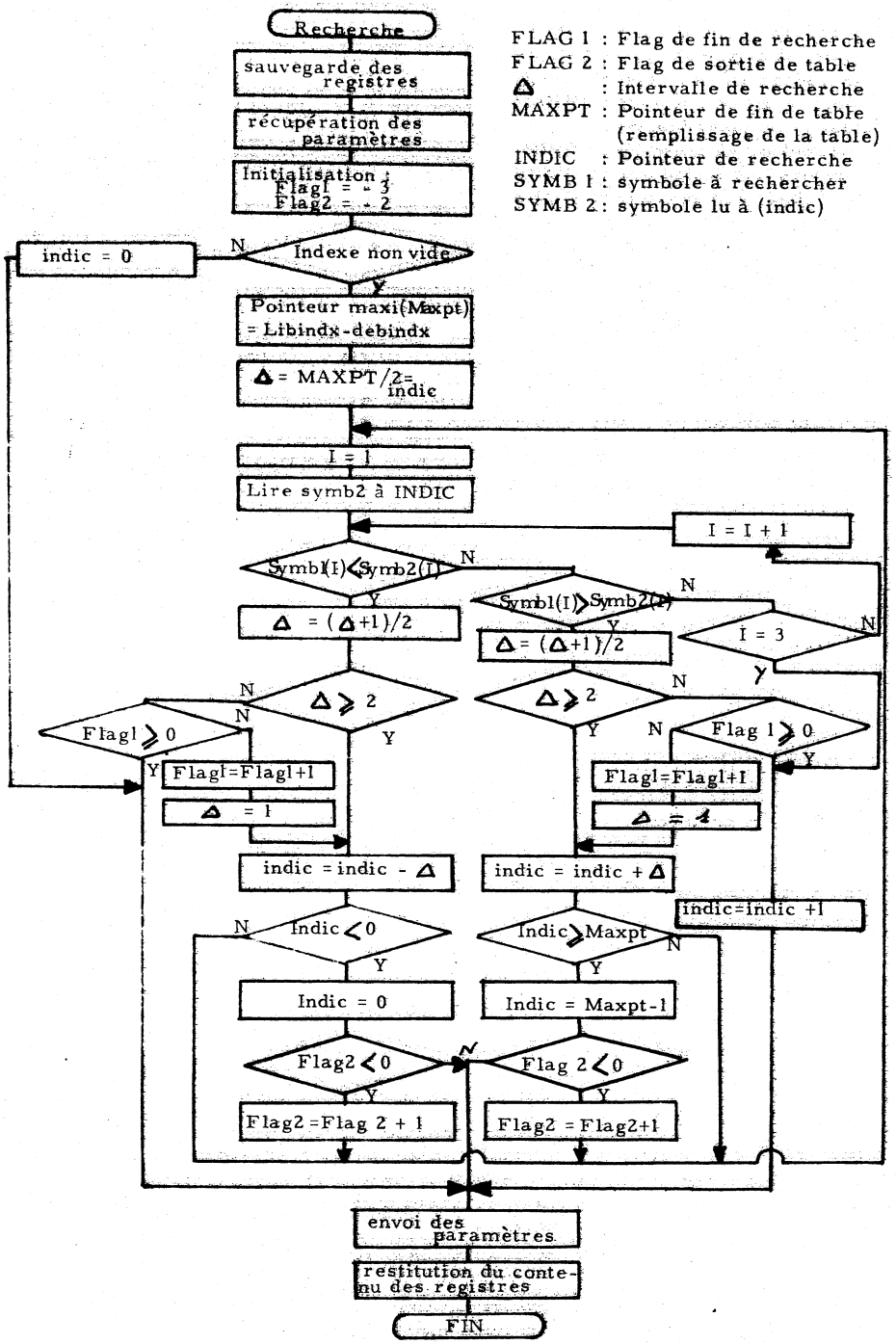


Fig.II.18 : Temps moyen de recherche dans l'index du fichier de définitions d'instructions, en fonction du nombre d'instructions contenues dans le fichier.

La courbe donnant le temps de recherche avec index en résident est tracée sachant qu'il faut une lecture des pointeurs de l'index et une lecture de l'index pour le mettre en résident d'autant plus longue qu'il y a de mnémoniques, ce qui donne une droite d'équation : 100 + 0,04 N (ms). La méthode de recherche avec traitement directement sur fichier est 3 fois moins performante en temps que la méthode avec mise de l'index en résident, pour un jeu d'instructions ne dépassant pas 200 instructions. Cependant c'est son utilisation qui a été retenue, sachant qu'elle ne fait pas intervenir de placemémoire et que le temps d'exécution n'est pas critique pour ce programme.

La figure II.19 donne l'organigramme du programme de recherche d'un mnémonique dans l'index du fichier, par cette méthode.



FLAG 1 : Flag de fin de recherche
FLAG 2 : Flag de sortie de table
Δ : Intervalle de recherche
MAXPT : Pointeur de fin de table (remplissage de la table)
INDIC : Pointeur de recherche
SYMB 1 : symbole à rechercher
SYMB 2 : symbole lu à (indic)

Fig. II.19 : Organigramme du sous-programme de recherche d'un symbôle dans l'index du fichier de définition d'instructions

5.4. Temps d'exécution moyen d'adjonction d'une instruction au fichier

Séquencement : (N : nombre d'instructions, n : nombre de mots)

- recherche			167 log N ms
- insertion	mise à jour	300 instructions Fortran	3 ms
		10 sous-programmes assembleur (+ appel)	8 ms
	décalage	2R 2W | File de $\frac{4N}{2}$ et $\frac{13N}{2}$ mots	$(50 + \frac{0,17N}{4}) \times 4$ ms
	adjonction	1 RFILE à 1 mot	50 ms
		100 instructions Fortran	1 ms
		2 WFile à 4 et 13 mots	100 ms
		1 RFile à 1 mot	50 ms
- mise à jour		1R 1W | File à 1 mot	100 ms

$$\overline{t_N} \# 512 + 0,17\,N + 167 \log N \text{ (ms)}$$

La figure II.20 donne le temps d'exécution moyen d'adjonction d'une instruction au fichier de définitions d'instructions, en fonction du nombre d'instructions déjà contenues dans le fichier.

5.5. Contrôle d'erreurs

5.5.1. Manipulation de fichiers

La création et l'effacement physique de fichiers ne peut être effectué qu'après avoir explicitement donné le nom ainsi que le type du fichier.

5.5.2. Manipulation de commandes :

Toute commande illégale n'est pas prise en compte et une liste des commandes existantes est imprimée.

5.5.3. Manipulation d'instructions :

Adjonction d'instruction au fichier :

Une instruction est prise en compte que si elle est correctement

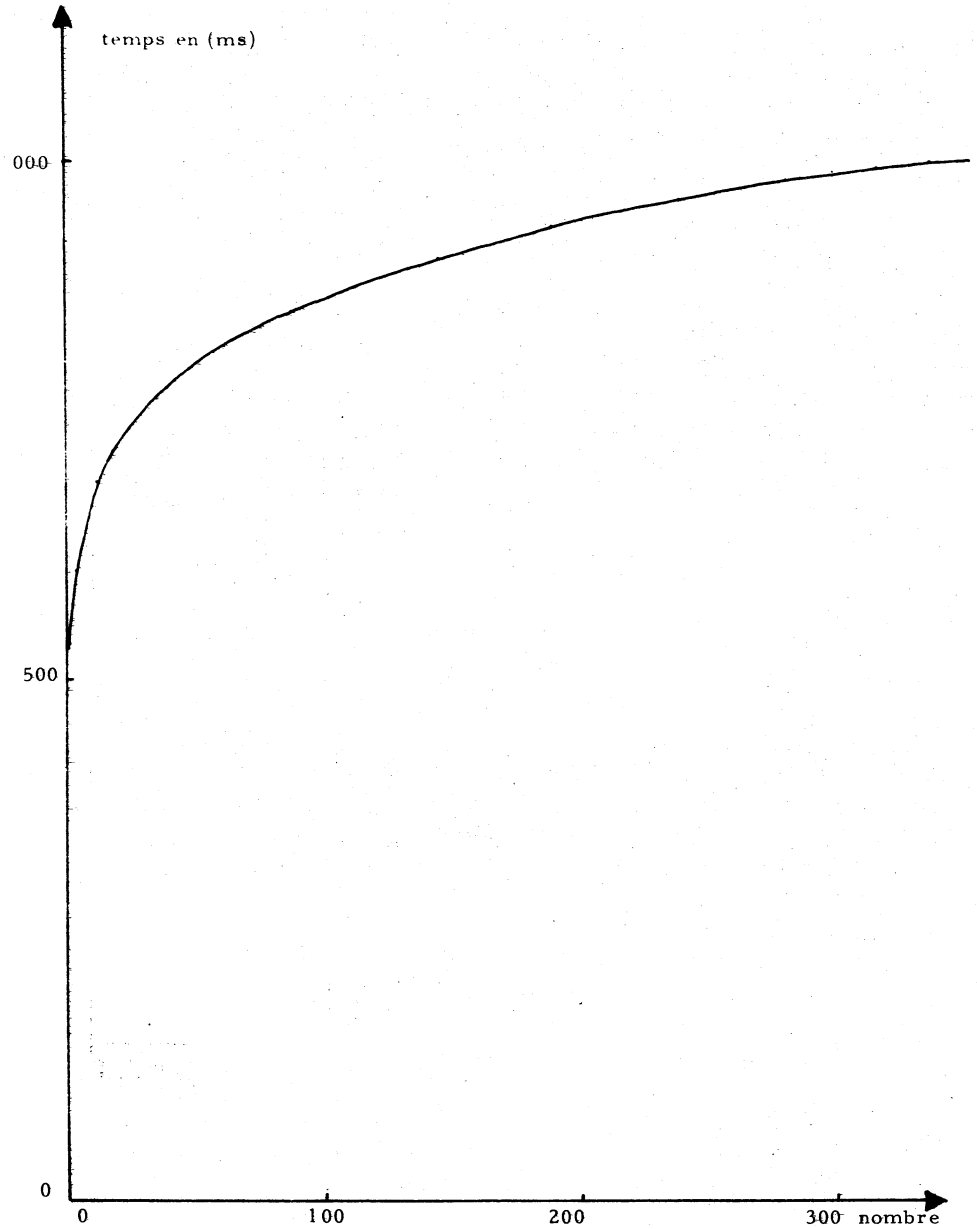


Fig. II. 20 : Temps d'exécution moyen d'une adjonction d'instruction au fichier de définition d'instructions

rentrée. Un contrôle est effectué sur les éléments suivants :

- instruction formée de 24 bits ou un multiple
- code opération contenu sur le nombre de bits indiqué
- égalité du nombre d'éléments des 2 listes de la définition d'instructions
- mnémonique sur 3 mots maximum
- syntaxe de la définition correcte
  - . seuls les caractères reconnus comme conformes doivent figurer dans l'expression de définition
  - . chaque élément de l'instruction doit être défini.

Suppression d'instruction du fichier :

- une instruction ne peut être supprimée qu'après une impression sur la console de visualisation

5.6. Format de rentrée d'une instruction sur console :

La structure adoptée est une structure de liste. Chaque liste étant séparée par des blancs. Les éléments à l'intérieur d'une liste sont séparés par des virgules.

Format

Code opération

␣␣ MNEMONIQUE ␣␣␣ n1,n2,.. ni,..n_n ␣␣ Cte , C(n), A(m), D(m), E(m), Cte

␣␣ : blancs ou pas

MNEMONIQUE : 6 caractères maximum

␣␣␣ : 1 blanc minimum

Cte : octale ou décimale

Cte : octale ou décimale

liste 1 : nombre de bits occupés par chaque élément de la liste 2 ni correspondant au ième élément de la liste 2

liste 2 : nature et numéro d'ordre dans la zône d'instruction.

mnémonique de l'instruction

remarque 1 : Si l'ordre des éléments dans l'instruction symbolique est le même que celui dans l'instruction en langage machine alors les éléments

de la liste 2 se succèdent de la façon suivante, dans la limite de leur existence :

- code opération
- C(1),...,C(i),...,C(n)
- |A;B;D;E| (1),...., |A, B, D, E| (i),..., |A, B, D, E| (m)
  ctes octales ou décimales

remarque 2 : une constante octale est repérée par une apostrophe devant le nombre (77)

remarque 3 : Si l'instruction ne peut être rentrée sur une seule ligne du terminale, un "slash" (/) doit terminer chaque ligne supplémentaire, à l'exception de la dernière ligne (terminée par "carriage return")

remarque 4 : le nombre de bits total occupé par l'instruction doit toujours être 24 bits ou un multiple entier de 24 bits.

V. ASSEMBLEUR

1. Introduction

Nous sommes maintenant en possession d'un jeu de définitions d'instructions sous forme codée, interprétable par un programme.

Nous allons concevoir un cross assembleur exécuté sur NORD-10 qui va traduire le langage symbolique GESPRO en langage machine.

Les caractéristiques principales de cet assembleur devront être une grande vitesse d'exécution et une grande sécurité d'assemblage. C'est principalement sous ces contraintes que va se définir la structure.

2. Généralités

La traduction des instructions, en langage symbolique, en instructions en langage machine une à une, n'est que l'aspect statique d'un assembleur. Il existe l'aspect interprétatif qui consiste à exécuter dans le programme source certaines pseudo-instructions et directives.

La part d'interprétation dans le mécanisme d'assemblage étant tout aussi importante que la part du traitement de substitution statique de base. La figure II.21 illustre schématiquement le processus d'assemblage.

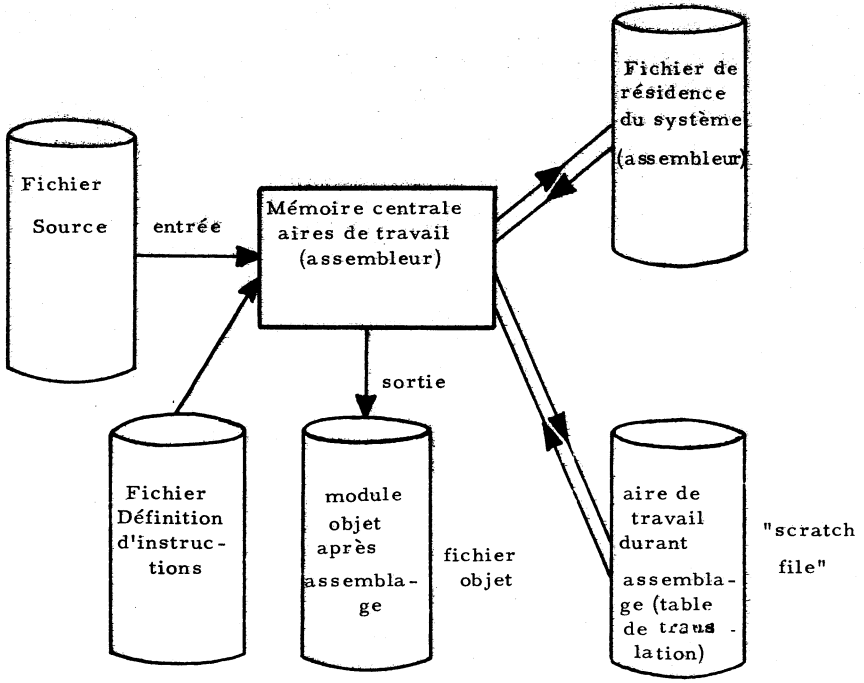


Fig. II.21 : Schéma de principe d'assemblage

Le travail principal d'un assembleur dans sa phase statique est de convertir les représentations symboliques des opérandes adresse et des mnémoniques code opération en forme binaire.

Cette conversion est faite par la table des symbôles, faisant l'équivalence entre un symbôle et sa valeur, ceci suppose qu'à la rencontre de chaque symbôle il faut rechercher dans la table s'il existe et si oui récupérer sa valeur, sinon l'introduire dans la table. Cette opération va se faire au moins une fois par instruction, d'où une structure de gestion de table du symbôle très élaboré si l'on veut atteindre de bonnes performances en temps.

3. Principe

3.1. Format général d'une instruction en format symbolique (fig. II. 22)

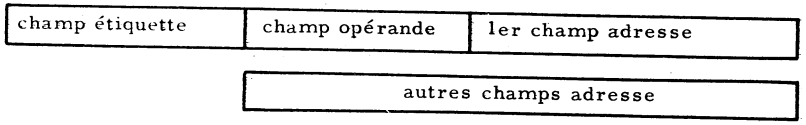


Fig. II. 22

3.2. Rappel du principe d'assemblage :

L'assemblage d'une instruction en langage symbolique va consister en la suite d'opérations suivantes :

a) décodage de l'instruction

- . entrée des symbôles, non encore définis en champ étiquette et champ adresse, dans la table des symbôles, avec leur définition et leur valeur pour les étiquettes.
- . entrée des constantes ($\geq$ 16 bits) dans une table
- . mise sous forme codifiée de l'instruction avec codage des éléments constitutifs de l'instruction suivis de leur valeur ou de l'information nécessaire, permettant de les retrouver dans les tables.

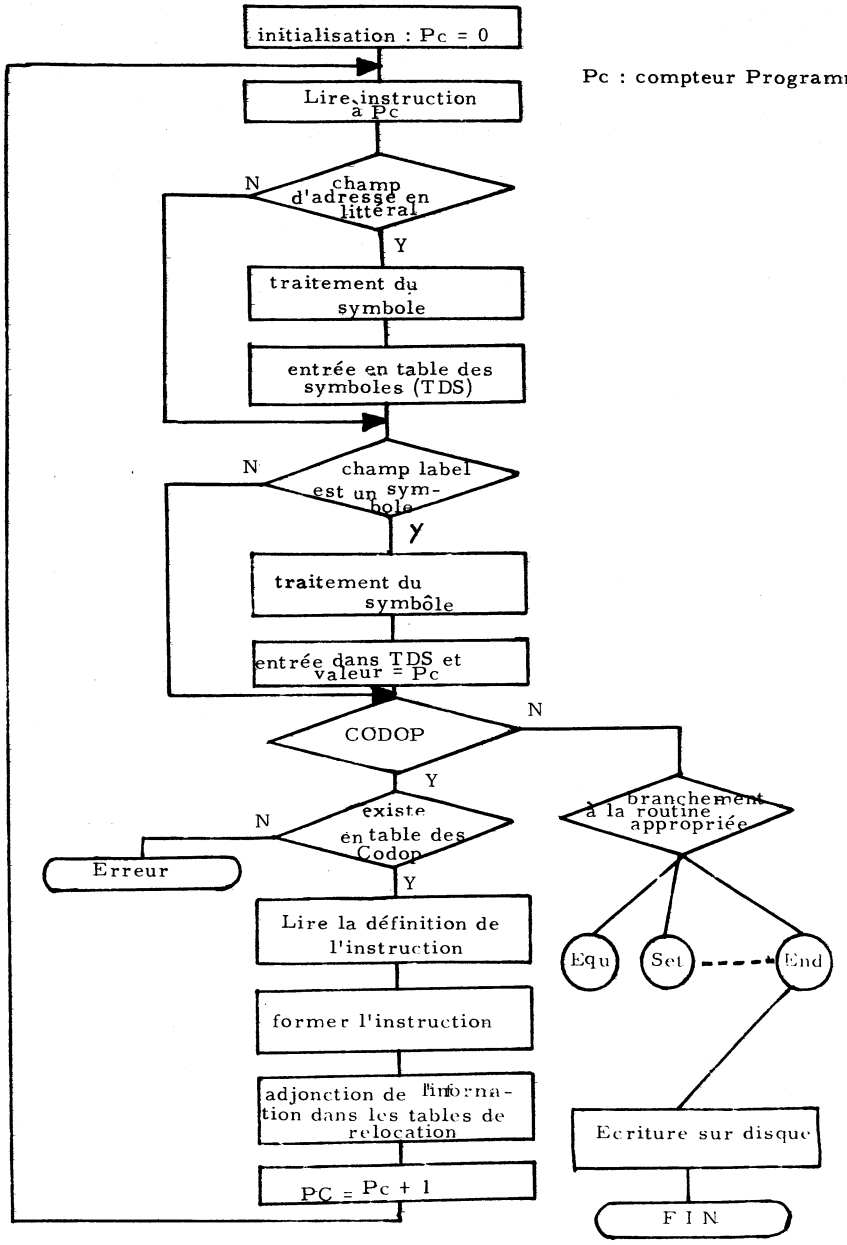
b) traitement de l'instruction en fonction de sa définition

- . constitution de l'instruction en format binaire relogeable avec placement des éléments définis et non translatables en objet.
- . placement des éléments non définis et translatables dans une table de relocation avec toute l'information nécessaire

ou traitement des directives :

- . débranchement à un sous-programme d'exécution

On trouvera à la figure II. 23 l'organigramme général du processus d'assemblage.



Pc : compteur Programme

Fig. II. 23 : Organigramme simplifié d'assemblage.

4. Support à l'assembleur et problèmes qui en résultent :

En tant que Cross-assembleur, le support principal de cet assembleur est le mini-ordinateur NORD-10 avec son système opérationnel SINTRAN III.

Pour obtenir un assemblage performant du point de vue vitesse d'exécution principalement, nous nous sommes définis la structure générale suivante :

- l'écriture du programme en langage assembleur NORD-10 exécuté sous SINTRAN III
- l'utilisation de fichiers NORD sans sous structure de façon à profiter au maximum des performances de gestion de fichiers de SINTRAN III. Utilisation au niveau source, objet et temporaire.

Problème :

NORD 10 est un mini-ordinateur à mot de 16 bits

GESPRO est un micro-ordinateur à mots de 24 bits

Solution adoptée :

4.1. Mot Gespro en mémoire Nord (figure II. 24)

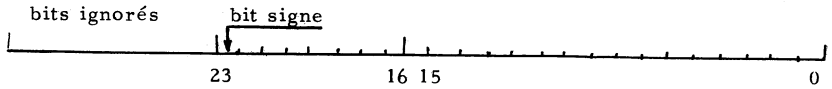


Fig. II. 24

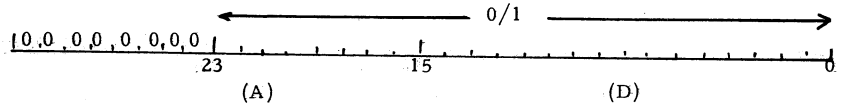
4.2. Gestion des constantes Gespro

Au niveau GESPRO, le logiciel dispose des constantes suivantes :

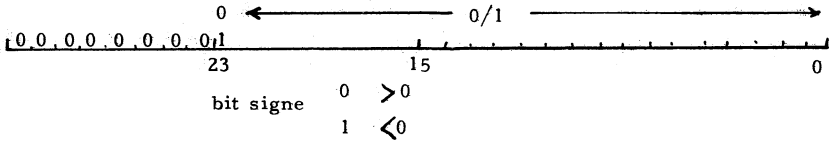
- constantes logiques sur 24 bits (non signées)
- constantes arithmétiques sur 24 bits signées (constante négative : bit 23 à 1)
- constantes arithmétiques < 24 bits non signées.

4.2.1 Traitement au niveau NORD-10 :

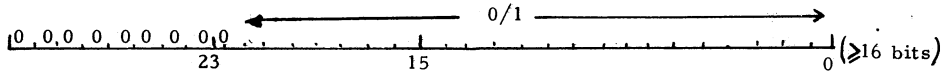
Constante logique :

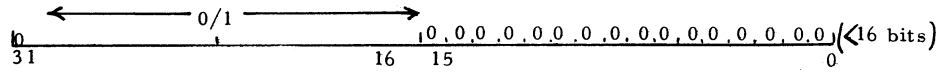


Constante arithmétique 24 bits signée :



Constantes arithmétiques <24 bits non signées





Remarque :

On appelle constante non signée, une constante devant laquelle ne doit pas se trouver un signe opérateur + ou - (unaire)

Ce qui n'exclu pas la différence de deux constantes non signées (a-b), pourvu qu'elles soient inférieures à 16 bits.

Au niveau traitement, la manipulation de : - b se fera comme un nombre négatif format NORD, soit bit 15 à 1.

Sous programme de gestion des constantes :

- conversion ASCII(décimal), binaire (octal)
  puisqu'il s'agit de constantes sur 24 bits, le traitement se fait sur double mot NORD-10
- Les autres conversions sont semblables à celles effectuées dans la gestion du fichier de définition d'instructions.

5. Pseudo-instructions et directives d'assemblage adoptées :

La reliabilité de modules programme est assurée par deux pseudo-instructions

DEF : relative aux symboles translatables, définis à l'intérieur d'un module, et qui sont utilisés dans d'autres modules.

REF : relative aux symbôles translatables utilisés à l'intérieur d'un module et qui sont définis dans d'autres modules

- pseudo-instructions de définition de métavariables :

SET : assigne une nouvelle valeur à une variable non translatable à chaque fois qu'elle est rencontrée.

EQU : assigne une seule valeur à une variable non translatable

- pseudo-instruction de gestion de constantes :

DATA : définit des constantes - logiques (24 bits)
- arithmétiques (24 bits)

- pseudo-instruction de gestion de zône mémoire :

RES : réserve le nombre de mots de 24 bits indiqué, avec une mise à zéro de ces mots. Chaque mot réservé figurera sur le "listing" objet.

Cette pseudo-instruction suivie de :

*+N : permet de réserver le nombre de mots indiqué par incrémentation du compteur programme de : N.
Les mots réservés ne figureront pas sur le "listing" objet.

- pseudo-instructions de début et fin de programme :

BEG : définit le point d'entrée d'un programme

END : . signale la fin d'un module programme
. précédée du caractère : $, signale la fin d'un programme à assembler.

- pseudo-instructions de contrôle du listage :

INF : (borne inférieure) arrête l'impression (ignorée à l'assemblage)

SUP : (borne supérieure), démarre l'impression (ignorée à l'assemblage).

Directives d'assemblage

- directives de branchement :

GOTO : effectue un débranchement lors de l'assemblage à une adresse spécifiée

IF : effectue un débranchement conditionnel.

Ces directives peuvent servir de base pour générer des modules d'acquisition spécialisés.

6. Différentes phases d'assemblage

6.1. Décodage

Chaque instruction symbolique est mise en format A1 dans un buffer puis décodée, pour donner une nouvelle instruction sous forme codée.

6.1.1. Codes utilisés

L'instruction mise sous forme codée, est composée de deux listes d'éléments. La première relative à la zône étiquette et code opération, la deuxième relative à la zône argument.

Les deux listes sont séparées par un élément séparateur assurant l'équivalence avec la définition d'instruction formée des deux zônes :

- zône commande
- zône argument

Chaque élément d'instruction (valeur ou indice de repèrage) est précédé d'un code. La figure II.25 explicite les codes utilisés.

6.1.2. Table des symbôles

Lors du décodage chaque symbôle différent rencontré en zône étiquette ou en zône argument doit être mémorisé dans une table avec sa valeur et sa définition. C'est le rôle de la table des symbôles.

La figure II.26 en donne la structure interne, relative à l'information ; avec les codes utilisés pour définir les symbôles rencontrés.

Code (octal)	Signification	
24	Séparateur de liste	
36	Double virgule (sert à ignorer un élément)	
50	Réservation de buffer (*+N)	
62	Fin de l'instruction	
0	Pas d'étiquette	zône étiquette
7	Symbole en étiquette	
11	Code opération	zône commande
13	Directive	
1	Simple constante 16 bits	Zône argument
2	Constante > 16 bits	
5	Symbôle en référence en avant	
6	Symbôle en référence extérieure	
7	Symbôle défini	
10	Symbôle défini comme métaétiquette	

Fig. II.25 : Codes utilisés au cours du décodage des instructions en format symbolique.

	SY	
Nom du symbôle	MB	
	OL	
valeur		
définitions particulières	1	ID, IE (définis par EQU)
	2	A, B, D, E (" " ")
	3	définition extérieure
	4	métaétiquette (GOTO)
	5	métaétiquette (1F)
définition de la variable	0	non définie
	2	référence extérieure
	4	définie par SET
	5	définie par EQU
	6	simple définition (représente une adresse)
	7	non définie (mais pas de listage d'indéfinition)

Fig. II. 26 : Structure interne de la table des symbôles

6.1.3. Initialisation de la table des symbôles :

Les métavariables à définir à l'initialisation sont les symbôles relatifs aux registres : A, B, D, E, ID, IE.

Ils sont introduits dans la table comme métavariables définies par EQU avec en définition particulière :

- le code 1 pour ID, IE
- le code 2 pour A, B, D, E

6.1.4. Autres tables associées :

Au cours du décodage, on rencontre en zône argument des instructions, des symbôles non encore définis (les références en avant).

En plus de l'entrée du symbôle en TDS, il faut garder en mémoire, jusqu'à l'entrée de l'information dans la table de translation, au cours du traitement de l'instruction codée, la valeur à ajouter à celle du symbôle pour constituer l'adresse. Ce sera le rôle de la table des références en avant, contenant en plus pour chaque valeur, l'adresse du symbôle correspondant en TDS (fig. II. 27).

En plus des références en avant, au cours du décodage dans les mêmes conditions nous allons rencontrer des références extérieures.

Ces symbôles ne seront définis qu'à l'édition de liens. Nous allons donc constituer une table (la table des références extérieures) qui servira de stockage de l'information jusqu'au chargement et donc figurera dans le fichier objet comme information de "relocation".

Il s'agit de garder en mémoire la valeur à ajouter à celle du symbôle pour constituer l'adresse, associée à l'information nécessaire pour retrouver le symbôle dans la table des définitions extérieures. C'est une recherche de symbôles par comparaison qui sera utilisée, d'où l'introduction du symbôle lui-même dans la table des références extérieures (fig. II. 28). En cours de décodage nous allons aussi rencontrer des constantes.

Celles inférieures à 16 bits pourront être introduites directement dans la zône de codage précédées d'un code. Le pas de cette zône étant 2 mots un pour le code, l'autre pour l'information, les constantes supérieures à 16 bits ne pourront y être placées directement. D'où un stockage temporaire, jusqu'au traitement de l'instruction codée, dans une table (la table des constantes) (fig. II. 29).

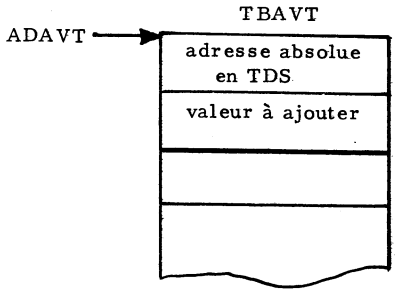


Fig. II. 27 : table des références en avant

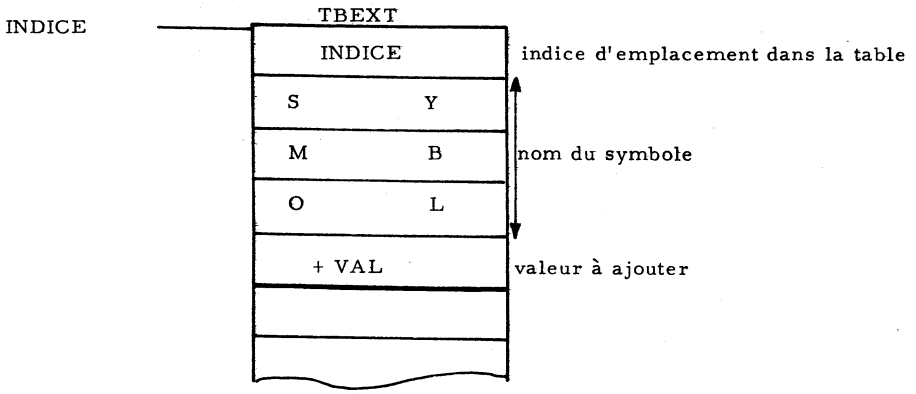


Fig. II. 28 : Table des références extérieures

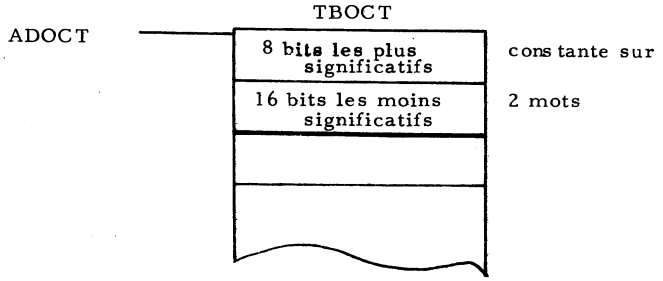


Fig. II. 29 : Table des constantes

Table des métaétiquettes relatives à : IF : (fig . II. 30)

Cette table contient le nom des métaétiquettes relatives à la directive IF, pour laquelle il y a eu saut à l'assemblage.

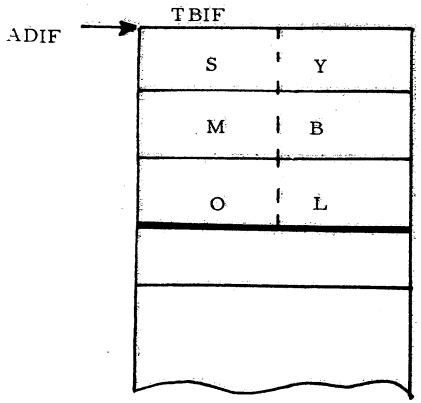


Fig. II.30

6.15. Recherche des erreurs :

La recherche d'erreurs au niveau du décodage doit permettre de détecter toute erreur de syntaxe au niveau de chaque instruction, au niveau de l'ensemble des instructions, et au niveau du programme lui même (instructions + directives).

au niveau instruction :

- syntaxe des symboles
- syntaxe des constantes
- existence du code opération ou directive
- syntaxe de la zône commande
- syntaxe de la zône argument
- syntaxe de chaque directive

au niveau ensemble d'instructions :

- symbôles en multi-définition

au niveau programme :

- existence des directives de début et fin de programme
- indéfinition de symbôles

d'autres contrôles au niveau assembleur doivent être faits :

- dépassement de capacité des tables
- "overflow" pour constantes et adresses

6.2. Traitement de l'instruction codée :

A partir de l'instruction sous forme codée et de la table de définition des instructions, nous allons élaborer l'instruction en format binaire relogeable constituée d'un ou plusieurs mots objet et de l'information nécessaire à la translation des adresses, mise dans une table (table de translation)

6.2.1. Organisation de la table de translation

Les instructions sont traitées en considérant que la première instruction exécutable du programme est à l'adresse : 0

Toutes les adresses seront donc relatives à : 0, début du programme.

Le chargement de ce programme se fera après translation des adresses de la valeur de l'adresse de chargement.

L'information relative à un symbôle dans la table doit permettre de placer en objet la valeur translatée donc :

- l'adresse à laquelle il se trouve dans le programme (valeur du compteur programme)
- le numéro dans l'instruction, du bit le plus significatif occupé.
- le nombre de bits occupés
- la valeur à translater

mais cette dernière n'est pas toujours définie à la rencontre de l'adresse. Un mot supplémentaire (une adresse) doit donc permettre de récupérer la valeur de la variable lors de la mise à jour en fin d'assemblage pour les références en avant et à l'édition de liens pour les références extérieures.

La figure II.31 donne la structure interne de la table de translation, pour les différents types de variables rencontrés.

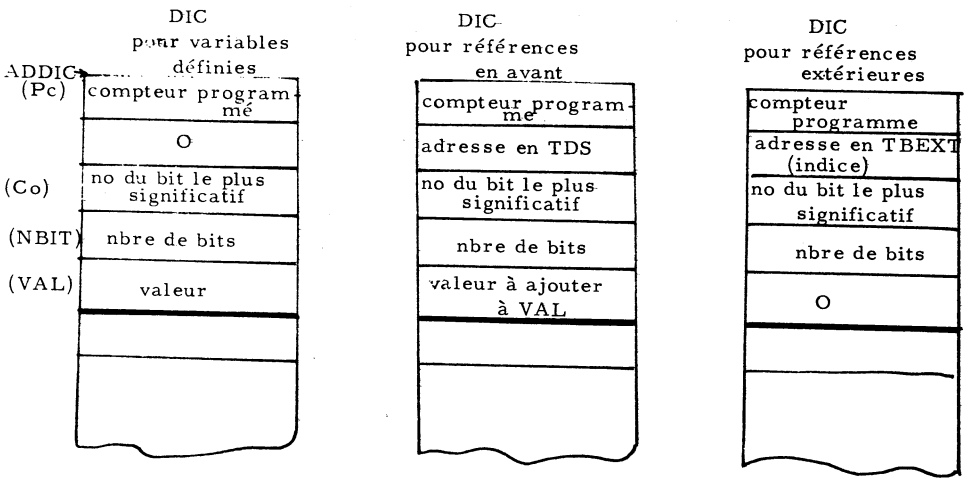


Fig. II. 31 : Organisation interne de la table de translation

afin de différencier les 3 sortes d'information.

Le 2ème mot pour les variables définies est mis à : 0.

Le 5ème mot pour les références extérieures est mis à : 0.

6.2.2. Placement des constantes et variables non translatables pour former le programme objet :

A partir de la valeur à placer, du nombre de bits occupés donné par la table de définitions d'instructions et de l'adresse dans le programme donné par le compteur programme, il s'agit d'élaborer un algorithme permettant de constituer un programme objet.

La méthode la plus simple sachant que l'assembleur NORD dispose de double mots, est de travailler sur un double mot avec un compteur de numéro de bit, le bit le plus significatif étant le bit 23 comme il a été convenu précédemment. L'organigramme du sous-programme de placement est détaillé à la figure II. 32.

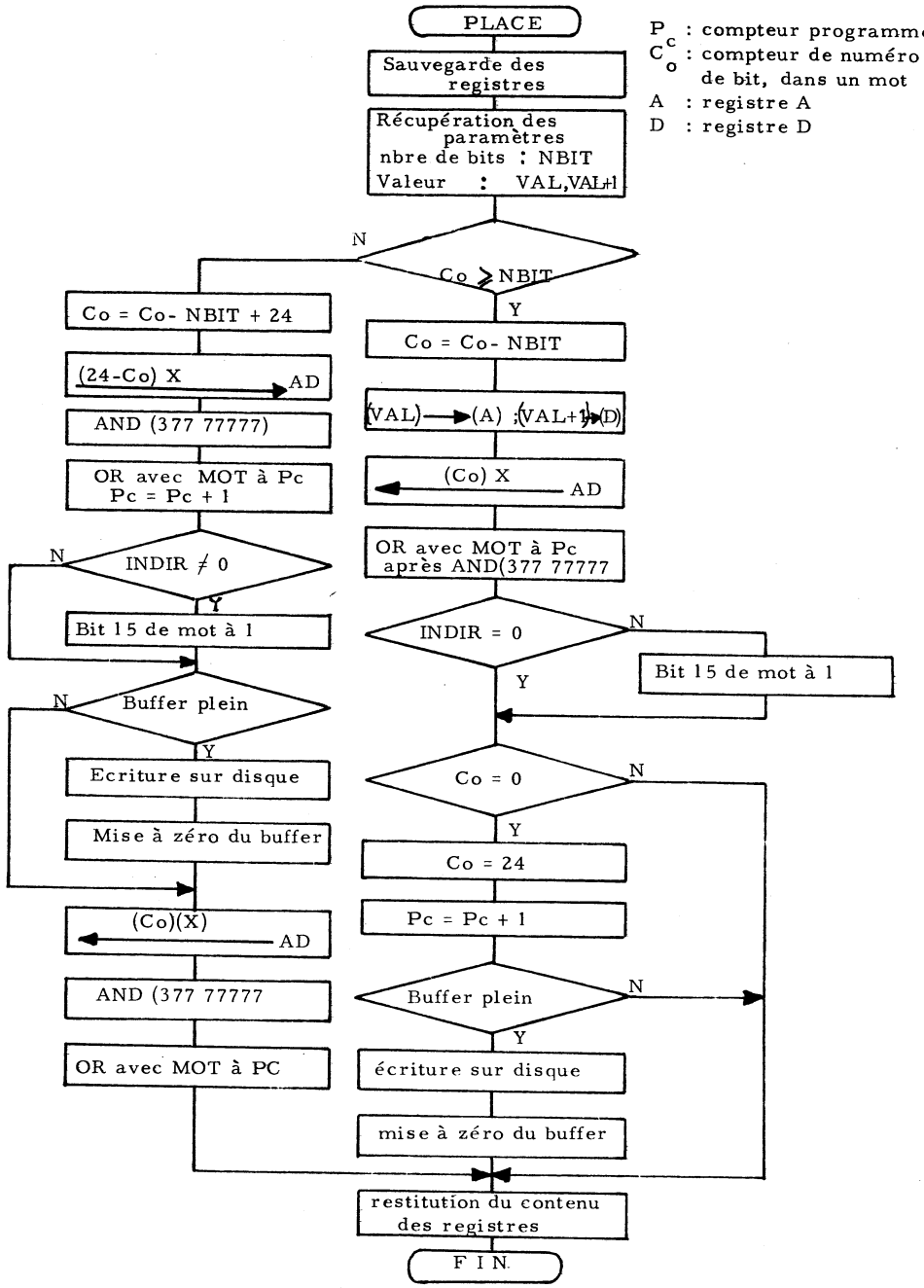


Fig. II. 32 : Organigramme du sous-programme de constitution de l'objet relogeable.

6.2.3. Détection d'erreurs :

A ce stade de l'assemblage, un contrôle doit être fait sur chaque élément constitutif des instructions.

On suppose qu'au départ il n'y a pas d'erreurs à la génération des instructions.

On doit contrôler tout d'abord si le nombre d'éléments en zône commande et en zône argument correspond au nombre de champs de la définition d'instruction . Ensuite on doit contrôler si chaque élément tient dans le champ qui lui est réservé.

En zône commande, on doit contrôler si chaque élément appartient à la liste des registres respectivement à chaque champ. En zône argument on doit contrôler que les adresses déclarées comme telles dans la définition ne sont pas négatives.

Si au cours d'un de ces contrôles on détecte une erreur un diagnostic explicite et spécialisé doit signaler l'erreur en question et tous les renseignements nécessaires à la correction de cette erreur. Les autres détections d'erreurs, doivent porter sur les débordements de table (table de translation principalement). Nous disposons aussi du contrôle d'erreurs effectué par SINTRAN.

7. Gestion de l'ensemble des tables

Maintenant que nous avons défini la nature des tables de travail, nous allons étudier leur organisation en mémoire et sur fichier, ainsi que leur organisation interne qui détermine le moyen de recherche. Cette étude est faite dans l'unique but, d'optimisation en temps et en occupation de place.

7.1. Organisation mémoire

7.1.1. Succession des adresses d'accès :

On distingue deux sortes de tables, celles à adresse d'accès aléatoire et celles à adresse d'accès continue.

table à adresse d'accès aléatoire :

table dont le rangement est sans enchaînement logique avec la succession des instructions du programme

- table des définitions d'instructions (faisant l'objet d'un rangement par ordre alphabétique)
* - table des symbôles (faisant l'objet d'une recherche non linéaire)
- table des définitions extérieures (dont le rangement se fait à la rencontre de la directive DEF)

tables dont l'accès se fait à partir d'une autre table :

* - table des symbôles
- table des références extérieures

} l'accès se fait par adressage à partir de la table de translation

tables à adresse d'accès continue :

- table source
- table objet
- table de translation
- tables temporaires relatives à une instruction
  - table des références en avant
  - table des constantes octales

Les tables à adresse d'accès continue pourront faire l'objet de stockage temporaire, partiel , sur fichier disque tandis que les tables à adresse d'accès aléatoire devront être mises en résident, ne pouvant être traitées par bloc comme les précédentes.

7.1.2. Evaluation de la dimension des tables

Ces dimensions se mesurent en fonction du nombre d'instructions du programme à assembler.

Seule la table de définitions d'instructions est fonction du jeu d'instructions considéré.

La figure II.33 donne le taux d'apparition des différents éléments d'instruction, rencontrés dans un programme GESPRO. Ce taux a été évalué en faisant une moyenne sur plusieurs programmes GESPRO de nature différente.

Nature de l'élément	facteur d'importance par symbole selon la nature	facteur de répétition par symbole	facteur de pondération = nbre de symboles différents par instruction	taux d'apparition résultant par instruction objet
Symbôles différents	1	1	1/4	1/4
Symbôles en référence en avant	1/3	3	1/4	1/4
Symbôles déjà définis	1/3	3	1/4	1/4
Symbôles en références extérieures	1/20	3	1/4	3/80
Symbôles en définitions extérieures	1/20	1	1/4	1/80
Métaétiquettes	1/50	1	1/4	1/200
Constantes 24 bits			1/4	1/20

Fig. II. 33 : Taux d'apparition moyen des différents éléments d'instruction dans un programme GESPRO

- Table de définition d'instructions :

Le jeu d'instructions utilisé contient 200 instructions. En moyenne chaque instruction occupera : 4 mots dans l'index et 13 mots en zône enregistrement soit au total 3,4 K mots.

Compte tenu de la perte de place dûe au partitionnement de la table

4 K mots

- Tableau de dimensionnement des tables (fig. II. 34)

Nature	nbre de mots /éléments	taux d'apparition /instruction objet	nombre de mots /instruction objet	32000 instructions objet
SOURC	15	1, 5	22, 5	340 K
OBJET	3	1	3	48 K
TDS	6 + X	1/4	$1,5 + \frac{X}{4}$	(24+4X) K
DIC (translat)	5	$\frac{1}{4} + \frac{1}{4}$	2, 5	40 K
TBEXT	5	3/80	$3/16 \simeq 1/5 (0,2)$	3, 2 K
TBENT	4	1/80	1/20 (0, 05)	0, 8 K
TBIF	3	1/200	3/200	0, 1 K
TBAVT	2	1/4	0, 5	
TBOCT	2	1/20	0, 1	

Fig. II. 34 : Dimensions moyennes théoriques des différentes tables de l'assembleur (X représente le nombre de mots supplémentaires pour le repérage de l'information dans la table des symboles)

7. 1. 3. Implantation des tables en mémoire :

Les tables traitables par bloc, ne vont figurer en mémoire de travail que bloc par bloc successivement dans un buffer. Il y a transfert d'un bloc du fichier disque dans le buffer à la lecture et du buffer sur fichier disque en fin de bloc, dans le cas de l'écriture.

C'est le cas des buffers : source, objet et information de translation. Les tables non traitables par bloc sont implantées entièrement en mémoire (TBIF, TBOCT, TVAVT, TBENT, TBEXT)

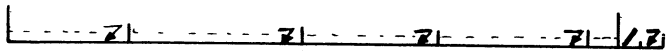
La table des symbôles, malgré son importante dimension (32 K maximum), doit être mise en résident totalement, si l'on veut garder de bonnes performances en temps, à cause de sa grande fréquence d'utilisation, et de son mode d'accès aléatoire.

a) dimensionnement des blocs et détection de fin de bloc

- buffer source

Le buffer est lu instruction par instruction, jusqu'au caractère "carriage return".

La fin du bloc coupant généralement une instruction, en deux parties le schéma suivant a été adopté :



buffer

un caractère "slash" suivi d'un "carriage return" occupe le 1er mot suivant le buffer.

A chaque lecture d'instruction, le slash est recherché, s'il existe c'est que l'on a atteint la fin du buffer. Le bloc suivant est alors lu ainsi que la fin de l'instruction.

Le temps moyen d'exécution de cette recherche à chaque instruction est de : 140 µs.

Si le temps de lecture sur disque d'un bloc est pratiquement indépendent de la dimension du bloc, sa fréquence de lecture en est inversement proportionnelle. Nous allons donc définir une dimension optimale du bloc. Sachant qu'en moyenne une instruction source occupe 15 mots mémoire, on en déduit la courbe d'équation : $\frac{15}{n}(50 + 0,01\,n)$ (fig. II. 35).

La dimension de 1 K mots est la valeur à partir de laquelle le temps d'exécution est suffisamment faible pour une occupation de place mémoire minimum ce qui donne un temps d'exécution par instruction de 0,9 ms

- buffer objet :

La même optimisation que précédemment peut être faite et c'est encore la valeur de 1 K mots qui est retenue. Bien qu'une valeur un peu plus petite dans ce cas soit acceptable, mais par souci de normalisation c'est cette première valeur qui est conservée.

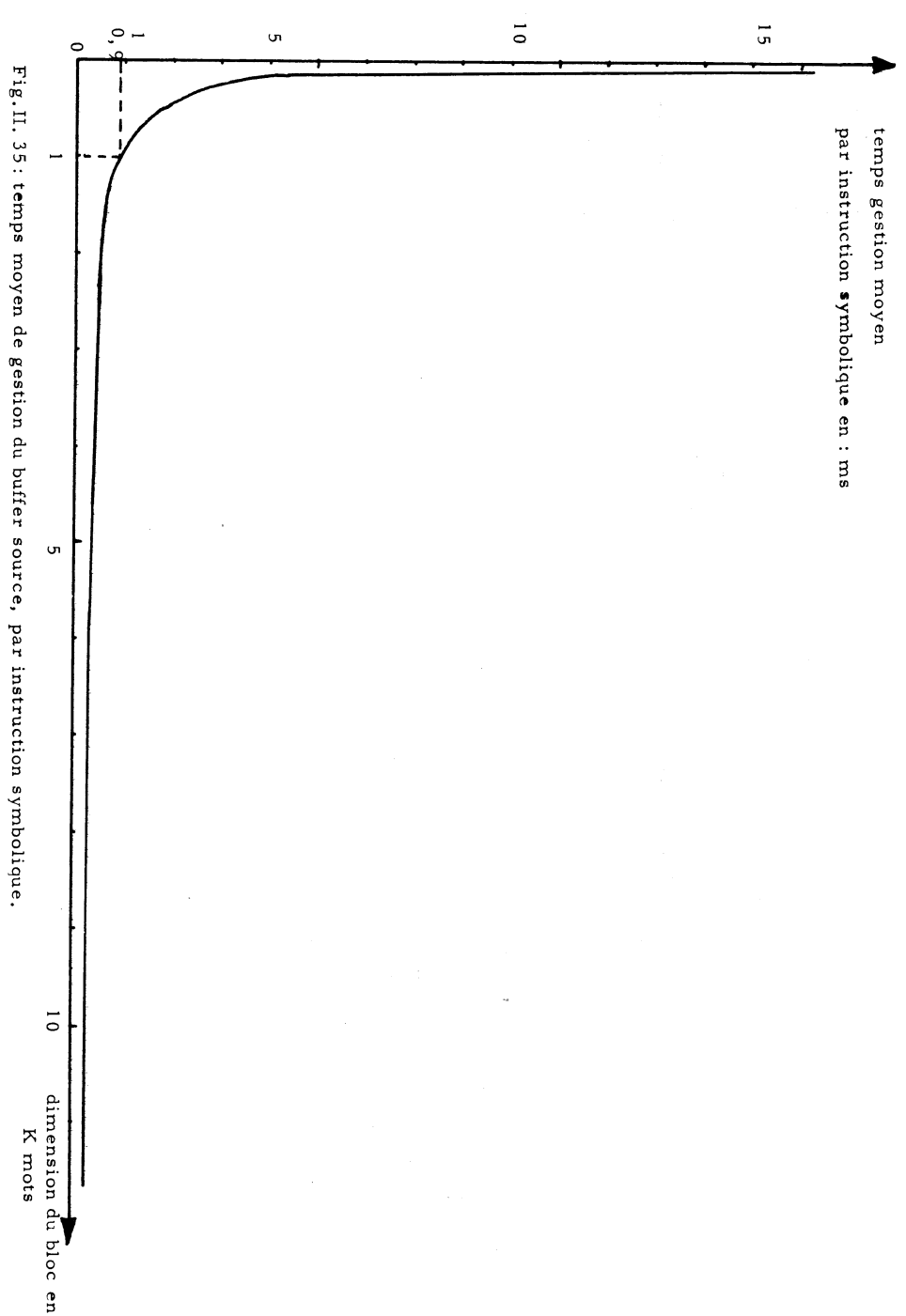


Fig. II. 35 : temps moyen de gestion du buffer source, par instruction symbolique.

Le temps d'exécution moyen par instruction sera 5 fois plus petit puisque les instructions occupent en moyenne 3 mots de 16 bits soit un temps de 0,18 ms/instruction.

La fin de bloc est détectée par un compteur et écrite dans le fichier objet relogeable.

- buffer de translation

La même dimension de bloc est retenue. Le temps d'exécution moyen est de : $0,9 \frac{2,5}{15} = 0,15$ ms/instr

La fin de bloc est détectée par un compteur. Mais contrairement aux cas précédents, le test de fin de bloc est effectué à chaque mot écrit, car la dimension de bloc n'est pas un multiple entier du nombre de mots (5 mots) par élément d'information. La table de translation est stockée temporairement jusqu'à la fin de l'assemblage, où elle est mise à jour, dans un fichier temporaire : le "scratch-file" de NORD. Ce fichier est toujours disponible et toujours ouvert pendant l'exécution d'un programme.

b) Image Mémoire

Les tables sont implantées en mémoire à la suite du programme principal et des sous-programmes assembleur. L'adresse de chaque table est fixée dans le programme à l'exception de la table des symbôles, qui est la dernière table et suit la table des définitions d'instruction, dont la dimension est fonction du jeu d'instruction.

L'adresse de la table des symbôles, contenue dans un mot, est fixée à l'initialisation en début d'assemblage.

L'image mémoire NORD, relative à l'assembleur sera donc constituée des programmes et sous-programmes, suivis des tables fixes, jusqu'à l'adresse de début de la table de définitions d'instructions.

La figure II.36 donne la map-mémoire de l'assembleur.

7.2. Recherche dans les tables à adresse d'accès aléatoire :

L'accès aux tables de références en avant et références extérieures se font par adressage, sans phase de recherche.

L'accès à la table des définitions extérieures se fait en recherche

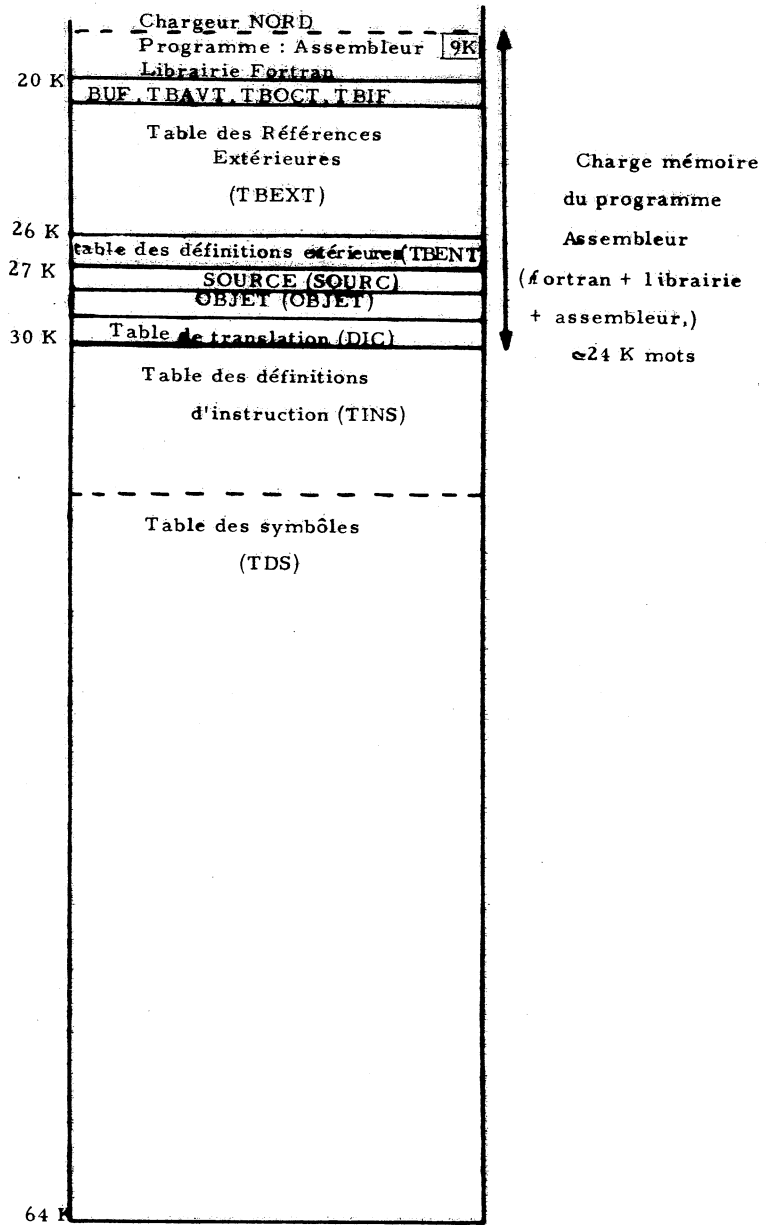


Fig.II. 36: Map Mémoire de l'Assembleur

Par comparaison de symbôles.

L'accès à la table de définitions d'instructions fait l'objet d'une recherche par dichotomie.

Accès à la table des symbôles

Au stade du traitement de l'instruction codée, la valeur de chaque symbôle défini en étiquette au décodage est introduite dans la table. Exception faite pour les métaétiquettes, qui sont ignorées après leur rencontre.

Si au décodage l'accès à la table au niveau du symbôle et à l'information associée était fait en recherche par comparaison de symbôle, après le décodage tout accès à la table est fait par adressage, ce qui supprime le temps de recherche généralement long.

L'organisation de la table doit donc être telle qu'au décodage on puisse avoir accès à partir du symbôle à l'information et par la suite à partir de l'information au symbôle (pour les indéfinitions en fin d'assemblage par exemple).

8. Organisation interne de la table des symbôles :

Au chargement on doit fournir au spécialiste système une image de l'occupation mémoire.

La façon la plus simple et la plus complète est de constituer un tableau avec les différents symbôles et leur valeur rangés par ordre alphabétique. Mais ce rangement nous interdit l'utilisation de la "hashing method".

Il nous faut donc étudier une structure à donner à la table pour optimiser la méthode de recherche logarithmique.

8.1. Différentes structures possibles :

Dans un rangement par ordre alphabétique, afin de garder la possibilité d'accès par adressage, il faut ajouter à chaque élément de table, un mot mémorisant la position initiale de chaque élément dans la table, comme le montre la figure II. 37.

adjonction d'un élément à l'adresse : 5.

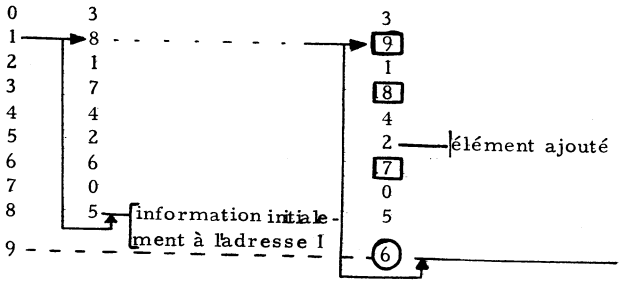


Fig. II. 37: Méthode de repérage dans la table des symbôles

Cette méthode nécessite à chaque insertion une translation d'une partie de la table, et une mise à jour d'une partie des indexes (entourés d'un carré sur la figure).

Boucles de programme :

```
BOUCL, SKP IF DB MGRE ST        (1,4 µs)
       JMP SUITE
       LDD  0,B                 2,9
       STD  7,B                 "
       LDD  2,B                 "
       STD  11,B                "
       LDD  4,B                 "
       STD  13,B                "
       AAB  - 6                 1,1
SUITE, LDT MAXPT                1,8
       LDA POINT                21,7 µs
       COPY SA DD

BUCL2, SKP IF DX MGRE ST        1,4
       JMP FIN
```

1 fois sur 2 en moyenne

```
       LDA  O,X                 1,8
       SKP IF DA MGRE SD        1,4
       JMP BUCL 2               1,8
       RINC DA                  1,1
       STA, OX                  1,8
       JMP BUCL 2               1,8
```

7,8 µs

Une perte de temps importante dans le rangement des symbôles par ordre alphabétique est dûe à la nécessité de translation d'une partie de la table à chaque insertion de symbôle.

Une solution pour diminuer ce temps de translation, consiste à ranger uniquement les symbôles par ordre alphabétique tandis que l'information liée aux symbôles fait l'objet d'un rangement successif ou continu.

L'équivalence entre symbôle et information correspondante, peut être réalisée par un mot relatif au symbôle attaché à l'information et un mot relatif à l'information attaché au symbôle, comme l'illustre la figure II.38.

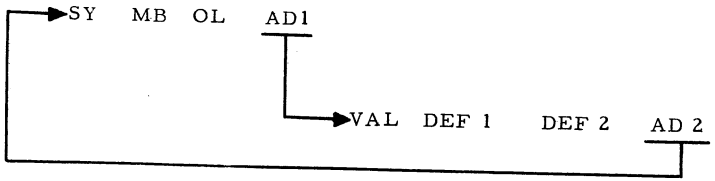


Fig.II. 38

Cette méthode limite la translation à quatre mots par élément, et nécessite une mise à jour à chaque insertion d'une partie des adresses AD2 seulement mais par contre entraîne une augmentation de 15 % de la dimension de la table.

Boucle de programme

BOUCL,	SKP IF DB MGRE ST	1,4 µs
	JMP SUITE	
	LDD 0,B	2,9 µs
	STD 10,B	"
	LDD 2,B	"
	STD 12,B	"
	COPY SD DX	1,1
	LDA 7,X	1,8
	AAA 10	1,1
	STA 7,X	1,8
	AAB - 10	1,1
	JMP BOUCL	1,8
SUITE,		21,7 µs

Le calcul du temps d'exécution est effectué en supposant qu'en moyenne les symbôles sont insérés en milieu de table.

Sachant que la table contient N symboles :

1ère méthode :

$$\overline{t}_N = 21,7 \frac{N}{2} + 7,8\ N$$

$$\boxed{\overline{t}_N = 18,65\ N (\mu s)}$$

2ème méthode :

$$\overline{t}_N = 21,7 \frac{N}{2}$$

$$\boxed{\overline{t}_N = 10,85\ N\ (\mu s)}$$

La deuxième méthode est 2 fois plus rapide que la première, pour une perte de place de 15 % c'est donc cette deuxième méthode qui va être mise en application.

8.2. Evaluation des temps d'exécution moyens de recherche de symbôles avec et sans adjonction dans la table

nombre de comparaisons moyen : $\frac{\log N}{0,3}$ lors d'une recherche

temps constant d'exécution d'éléments du programme de gestion de la table

45 instructions à 1,5 µs ⟶ 70 µs

temps d'exécution moyen d'une boucle de recherche :

30 instructions à 1,5 µs ⟶ 50 µs

temps d'exécution moyen d'une recherche avec adjonction du symbôle

$$\overline{t}_N = 70 + \frac{\log N}{0,3} \times 50 + 11\ N \quad (\mu s)$$

$$\boxed{\overline{t}_{N\ (ms)} = 0,07 + 0,166 \log N + 0,011\ N}$$

temps d'exécution moyen d'une recherche sans adjonction de symbole

$$\boxed{\overline{\overline{t}}_{N(ms)} = 0,07 + 0,166 \log N}$$

Ces temps d'exécution sont représentés graphiquement sur la figure II. 39.

8.3. Sous-programme de recherche dans la table des symbôles :

La partie de recherche générale de ce sous-programme est identique au sous-programme de recherche dans la table de définition d'instructions (figure II.19) avec en plus une partie ou fichier destinée à l'insertion des symbôles et l'information associée dans la table.

9. Exécution de la fin d'assemblage

9.1. Mise à jour des tables de translation et de définitions extérieures :

A la rencontre de la directive : END, précédé du caractère : $ et du nom du programme, l'assemblage est terminé. L'objet non translatable a été constitué et écrit sur le fichier objet. Par contre tous les symbôles rencontrés dans le programme avant qu'ils aient été définis, doivent faire l'objet d'une mise à jour. Il s'agit des symbôles en référence en avant et des symbôles rencontrés dans les directives : DEF .

9.1.1. Mise à jour de TBENT : (table de définitions extérieures)

A partir de l'adresse en table des symbôles, la valeur de chaque symbôle est introduite dans la table : TBENT.

En même temps un contrôle de définition de symbôle est effectué. Chaque indéfinition est suivie d'une impression du symbôle.

9.1.2. Mise à jour de DIC (table de translation)

A partir de l'adresse en table des symbôles, la valeur de chaque symbôle en référence en avant est introduite dans la table de translation après addition avec la constante à ajouter.

Un contrôle de définition est fait à chaque symbôle. Toute indéfinition est suivie d'une impression de symbôle avec son adresse dans le programme.

9.2. Table de symbôles et leur valeur :

Afin d'apporter une aide au spécialiste système, pour l'élaboration des programmes et leur implantation en mémoire, on doit lui fournir une visualisation de l'image de chaque programme après l'assemblage dans le but

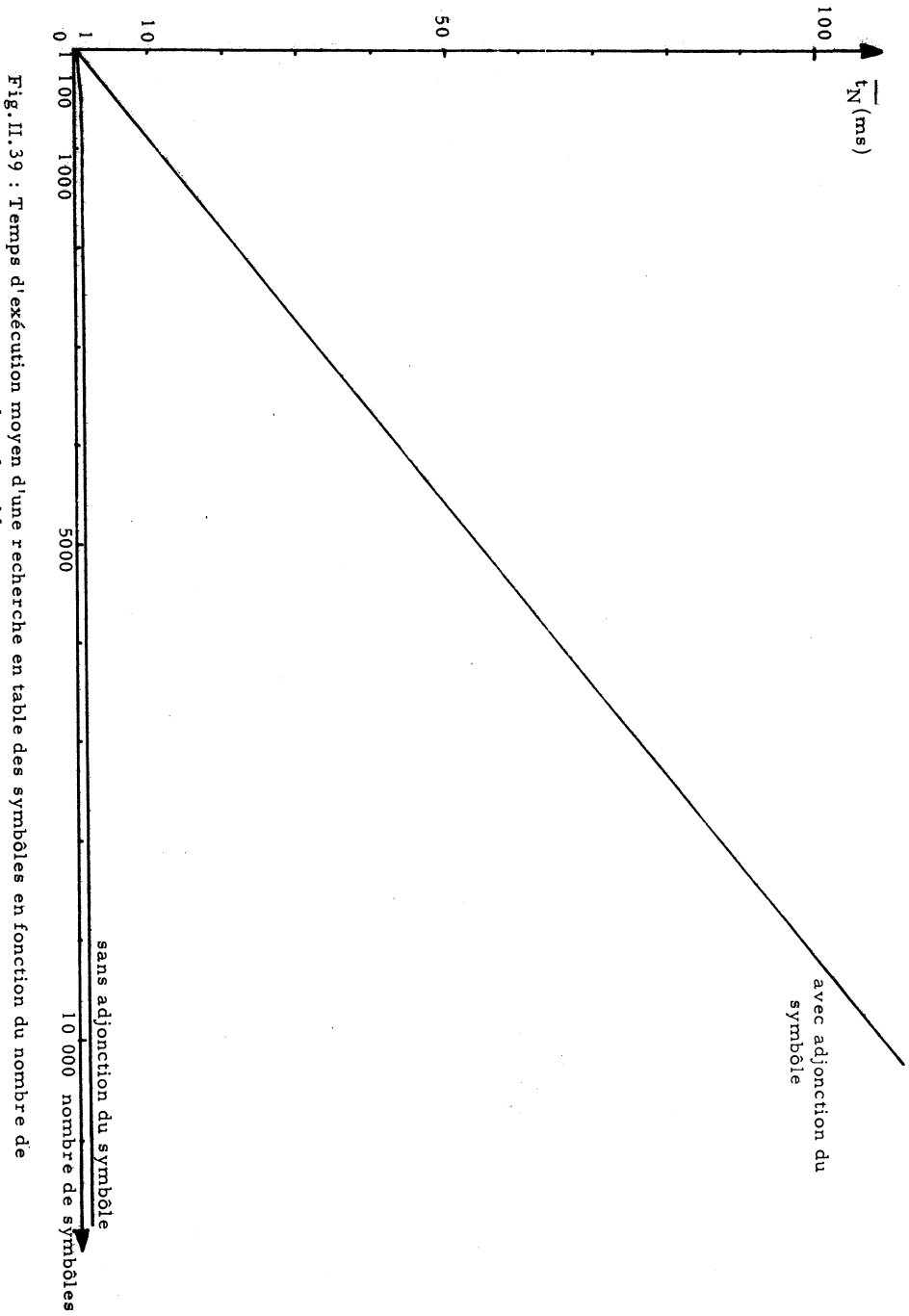


Fig.II.39 : Temps d'exécution moyen d'une recherche en table des symbôles en fonction du nombre de symbôles contenus dans la table.

de créer une visualisation de l'image mémoire au chargement.

Comme nous l'avons vu dans l'étude de l'organisation de la table des symbôles, la solution la plus adéquate et la plus rapide est une visualisation sous forme de tableau avec tous les symbôles translatables et non translatables, par ordre alphabétique avec leur valeur.

Cette table est constituée directement à partir de la table des symbôles en faisant un contrôle de définition et une différentiation des symbôles translatables par rapport aux symbôles non translatables, en mettant le bit 15 du mot de valeur à 1.

9.3. Organisation du fichier objet :

En fin d'exécution de la directive : END, les différentes tables sont écrites dans le fichier, ainsi que l'entête du fichier constituée du nom du programme avec les pointeurs des tables, le point d'entrée et le nombre d'erreurs à l'assemblage. Un exemple de fichier objet est présenté à la figure II.40.

10. Listings

En cours d'assemblage il y a impression ou non de l'objet. Avant chaque listing il y a impression de la date et du nom du fichier source. A la fin de l'assemblage il y a impression du nombre d'erreurs et une demande d'impression des tables, et si oui impression des tables figurant dans l'objet. (ANNEXE C).

11. Temps d'assemblage moyen d'une instruction

11.1. Initialisation (dans programme principal : Fortran)

400 instructions Fortran	4 ms
3 Open-file	300 ms

304/N

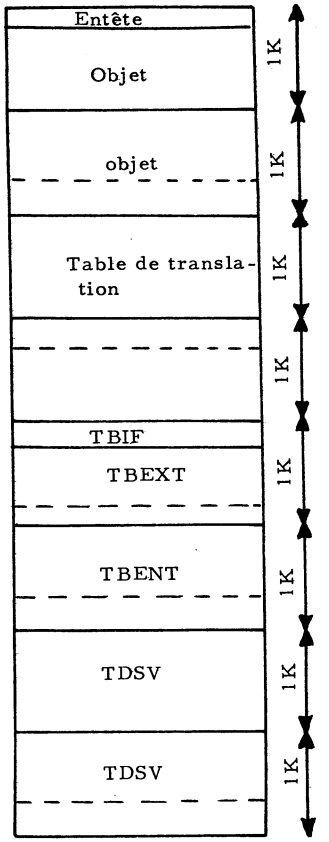


Fig. II.40 : Exemple de contenu du fichier objet relogeable

11.2. Initialisation (dans sous-programme principal assembleur)

4 SETBS	10 ms
2 RMAX	20
1 RFILE (Source) (1Kmots)	60
1 RFILE (INSD) (≃5 K mots)	100
200 instructions assembleur	0,4
mise à zéro (TDS, DIC, OBJET) (30K mots x 6µs)	180

370/N

11.3. Décodage

Lecture d'une instruction source	0,9 ms
accès à un sous-programme de gestion des constantes	0,3
2 accès à latabl e des symbôles avec en moyenne 1/4 d'adjonction	$0,14 + 0,33 \log N + \frac{0,011}{2} N$
1 accès à la table des directives	0,23
2 accès au sous-programme de gestion des caractères	0,3
500 instructions assembleur	1
	$2,9 + 0,33 \log N + 0,0028 N$ ms

11.4. Traitement de l'instruction

Ecriture de l'objet sur disque/instruction	$\frac{60 \text{ mx} \times 3}{1024}$	0,18 ms
3 accès au sous-programme "PLACE"	0,4	
Ecriture de la table de translation sur disque par instruction	$\frac{60 \times 2,5}{1024}$	0,15
500 instructions assembleur	1	
	1,75 ms	

11.5 Traitement de la directive END :

Mise à jour TBENT/instruction		négligeable
Mise à jour DIC/instruction (lecture sur disque + traitement)	DIC	$\frac{60 \times 2,5}{1024} + 0,02 \simeq 0,17$ ms
Constitution du fichier objet par instruction	OBJET	$\frac{60 \times (3 + 2,5)}{1024} \simeq 0,33$
		0,50 ms
100 instructions assembleur		0,2 ms
4 écritures sur disque	entête du fichier 220 tables fixes	$\frac{220 \text{ ms}}{N}$

Bilan :

$$\frac{900}{N} + 6 + 0,33 \log N + 0,0028 N$$

Ce temps d'exécution moyen d'une instruction est représenté graphiquement sur la figure II.41.

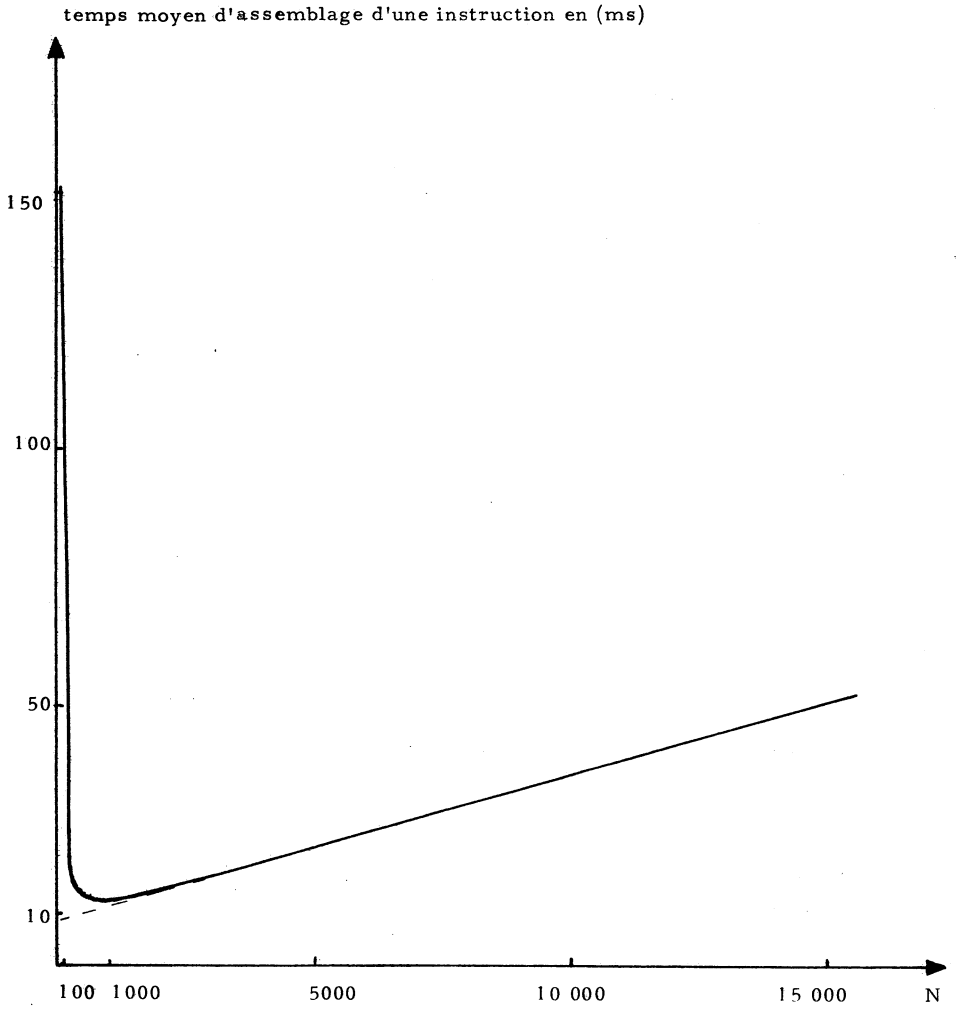


Fig. II. 41 : temps d'assemblage moyen théorique d'une instruction en fonction de la dimension du programme à assembler.

VI. CHARGEUR

1. Introduction

Le chargeur représente la phase finale du système de développement de programmes. A partir des modules objets créés lors de l'assemblage, il doit permettre de constituer physiquement un programme exécutable en mémoire.

Il est composé des fonctions principales rencontrées dans tout chargeur relogeable :

- relier les différents modules constituants d'un programme, pour en faire un module relogeable.
- reloger le programme en fonction de l'adresse de chargement
- placer physiquement le programme en mémoire

à ces fonctions doivent s'ajouter des fonctions spécifiques à GESPRO :

- contrôle du STATUS de GESPRO avant chaque chargement
- système de recherche d'erreurs élaboré
- contrôle et visualisation de l'implantation du programme en mémoire
- option de "debugging" sur les programmes chargés.

2. Principe

GESPRO se trouve dans un contexte "CAMAC" le chargement sera donc réalisé par des ordres CAMAC. Une librairie fournit une routine d'exécution d'ordres appelables par Fortran. Le Fortran dispose aussi des entrées sorties CAMAC. D'autre part l'optimisation en vitesse n'étant pas importante du fait du petit nombre d'opérations effectuées par ce programme, le programme principal sera donc écrit en Fortran.

La structure sera : le programme principal en Fortran et les sous-programmes en assembleur.

Le chargeur est composé des phases principales suivantes :

- édition de liens des modules relogeables
- translation du programme relogeable
- chargement du programme relogé avec possibilité d'impression

La figure II. 42 illustre le processus de chargement d'un programme.

Pour ces 3 phases nous devons avoir accès à chaque fichier objet d'où la nécessité lors de la rentrée des noms des fichiers source et objet, de les mémoriser dans des tables. A chaque utilisation ultérieure, les fichiers sont ouverts puis refermés successivement. Cette méthode autorise l'utilisation successive d'un même fichier objet pour plusieurs fichiers source.

Nous allons étudier dans les paragraphes suivants le développement de chaque partie du chargeur.

3. Structure adoptée :

3. 1. Editeur de liens

Le lien entre chaque module au niveau assembleur est réalisé par les directives : EXT et DEF faisant l'objet des tables de définitions extérieures et des références extérieures.

Au niveau chargeur à partir de chaque élément de la table des références extérieures il s'agit de rechercher la valeur du symbôle dans l'ensemble des tables de définitions extérieures relatives à un programme.

Dans ce but le premier travail de l'éditeur de liens est de rassembler toutes les tables de définitions extérieures en une seule avec une structure adaptée à la recherche de symbôles.

La solution de rangement par ordre alphabétique des symbôles peut cette fois encore s'appliquer. Cette méthode va permettre de détecter facilement une multi-définition de symbôle dans un programme.
Le sous-programme utilisé est équivalent à celui rencontré dans l'assembleur pour la gestion de la table des symbôles.

Cette table constituée, l'édition de liens peut être faite par la mise à jour de toutes les tables de translation relatives au programme (voir figure II.43) Le résultat est un programme relogeable.

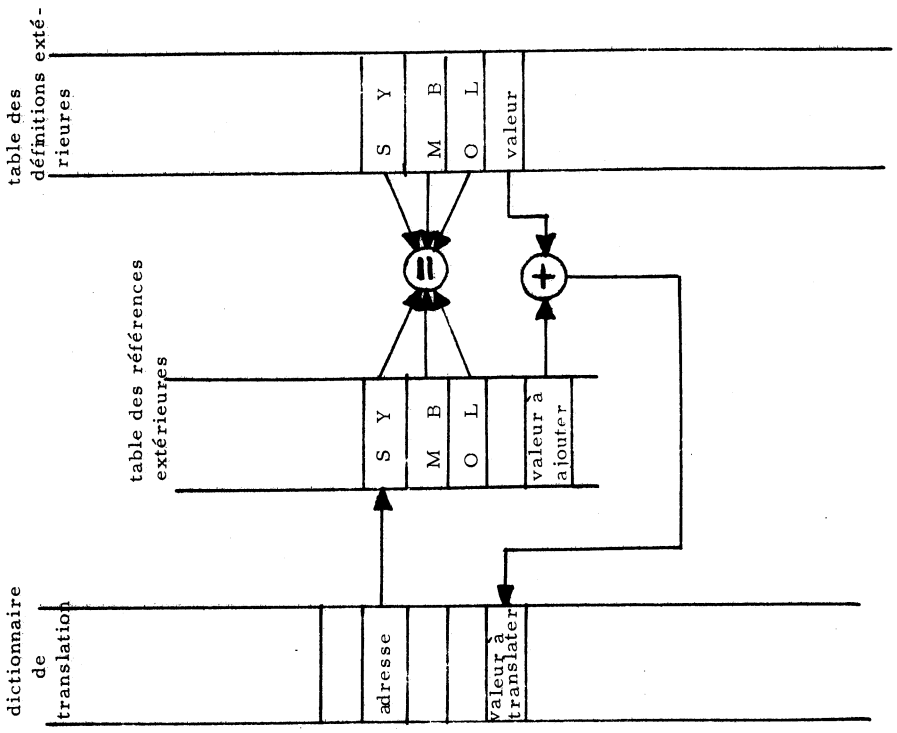


Fig. II. 43 : Schéma de principe d'édition de liens

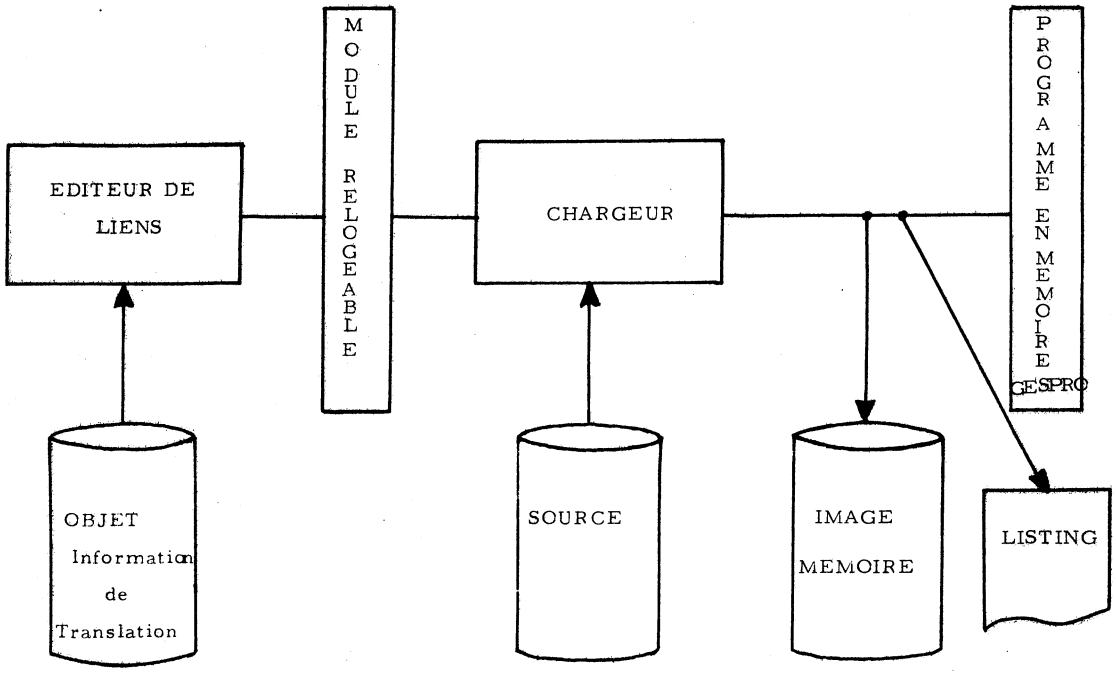


Fig. II. 42 : Schéma de principe général du chargeur

3.1.1. Recherche d'erreurs :

Un contrôle de multidéfinition est effectué lors de la constitution de la table des définitions extérieures

- un contrôle d'indéfinition est effectué à l'édition de liens, par détection d'une valeur nulle dans la table des définitions extérieures. A chaque erreur un diagnostic est imprimé.

3.2. Translation

La création d'un module chargeable à partir des modules relogeables, peut se faire de deux manières :

- en chargement automatique, à partir d'une adresse initiale, les modules constituants du programme s'enchaînant de manière continue.
- en chargement module par module, l'image mémoire résultante peut être discontinue. Le plus petit élément continu étant un module. Cette deuxième méthode permet de laisser libre des zônes mémoires fixes utilisées directement par le "hardware".

Le placement des adresses translatées dans chaque module est assuré par un sous programme "PLACE", équivalent à celui utilisé dans l'assembleur. La figure II.44 présente l'organigramme du sous-programme de translation des modules relogeables.

3.3. Chargement :

Le chargement de GESPRO est fait mot par mot après un test de son Status. Après chargement de chaque module il y a création d'une image mémoire, si elle a été demandée. La figure II.45 présente l'organigramme du sous-programme de chargement.

3.3.1. Création d'une image mémoire :

L'image mémoire est introduite dans un fichier dans lequel on ne trouve que le programme en binaire, 2 mots de 16 bits pour un mot 24 bits ; avec en entête sur les 4 premiers mots :

- le point d'entrée
- l'adresse haute
- l'adresse basse
- le nombre de mots de 24 bits constituants le programme

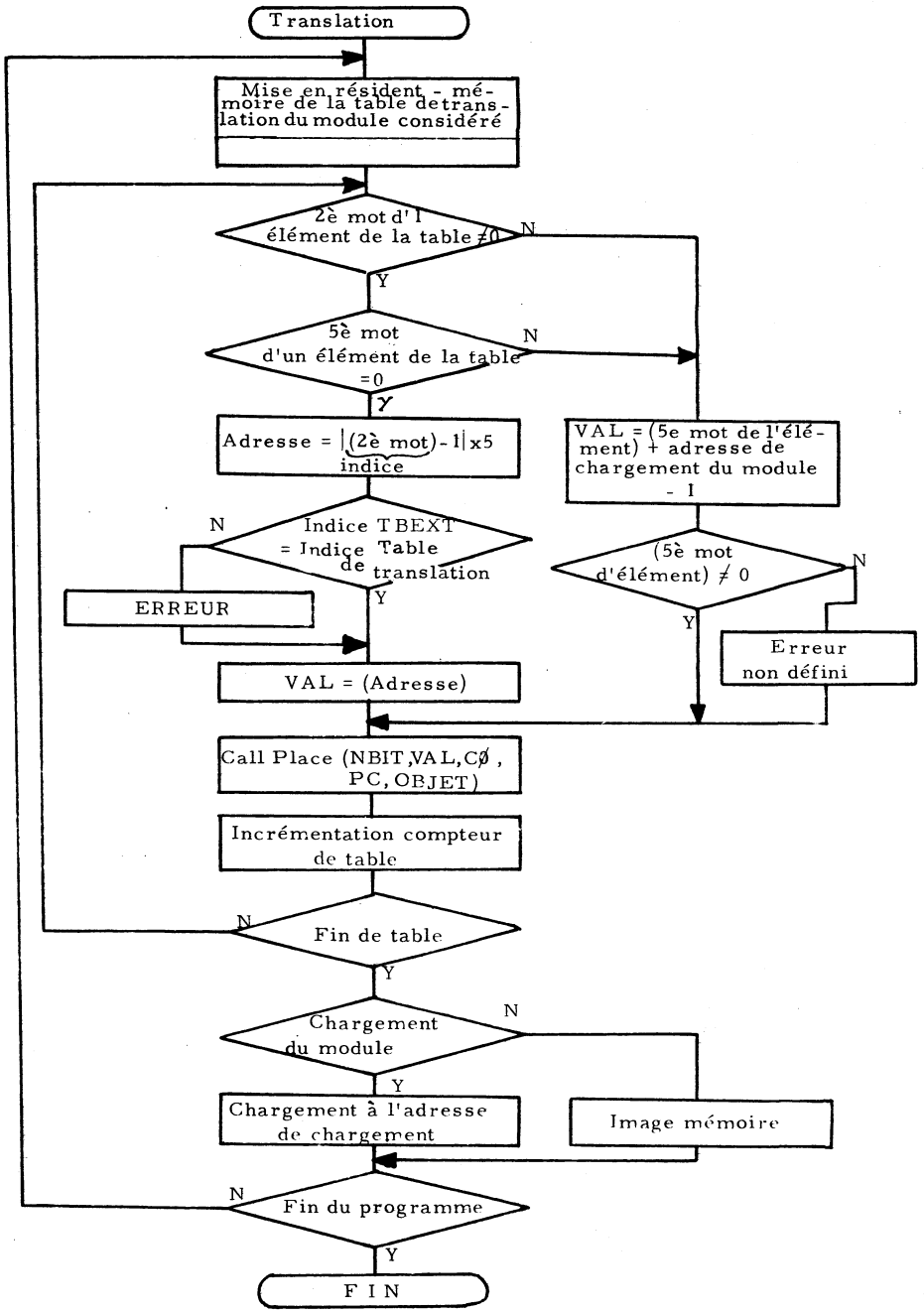


Fig. II. 44 : Organigramme du sous-programme de translation des modules relogeables.

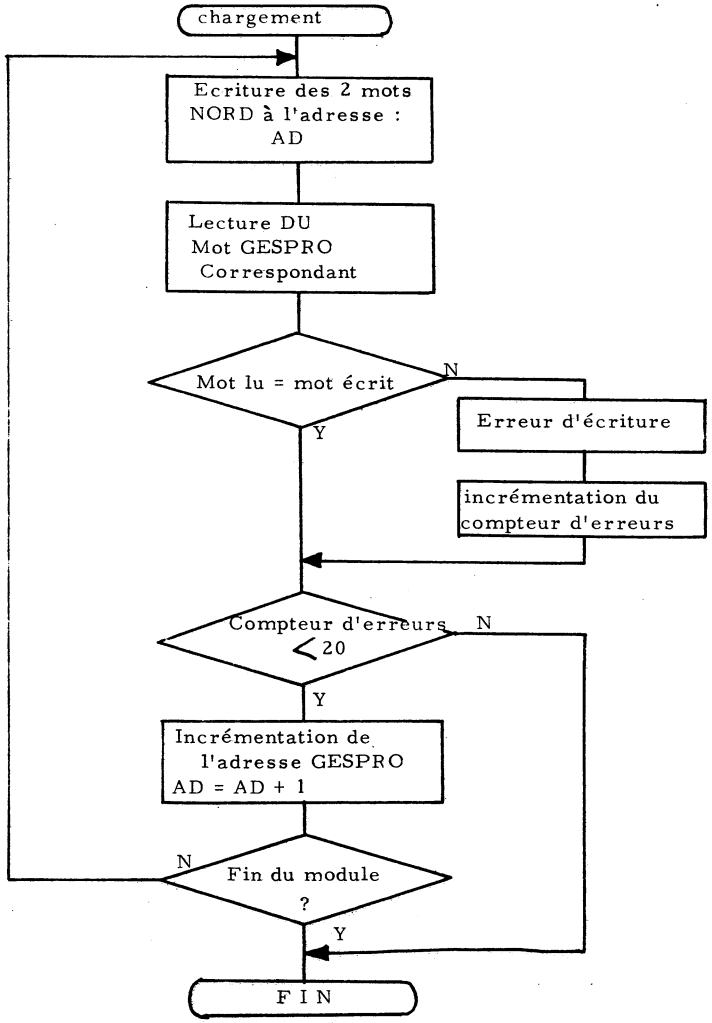


Fig. II. 45 : Organigramme du sous-programme de chargement

3.3.2. Contrôle d'implantation des programmes en mémoire

Ce contrôle est réalisé sous forme de MAP-Mémoire, vérifiant en fin de chargement s'il n'y a pas chevauchement des modules chargés. Tout chevauchement est signalé par un message d'erreur avec les adresses en question. Un contrôle d'occupation de la case mémoire $(3777)_8$ (JMP WAIT) est effectué s'il y a occupation de cette case par le programme utilisateur, un message est imprimé.

3.3.3. Visualisation de l'image mémoire

Après chargement de chaque module, un tableau de toutes les variables par ordre alphabétique et leur valeur est imprimé à la demande de l'opérateur. (ANNEXE D).

3.4. Lancement de GESPRO (figure II. 46)

3.5. Option "debugging"

- l'option "list" consiste à donner la possibilité à l'opérateur de sortir sur une terminal au choix une zône mémoire désirée
- l'option "core" permet d'imprimer mot par mot une zône mémoire désirée avec la possibilité après chaque impression de modifier la case mémoire en question.
- l'option "write memory" permet d'écrire automatiquement dans une zône mémoire désirée, une suite de mots à partir d'un mot initial et d'un facteur d'incrémentation
- l'option "halt" permet l'arrêt de GESPRO par arrêt de l'horloge
- l'option "wait" permet de tester si l'ordre "start" a déjà été donné, si non on le signale à l'opérateur par l'impression de "GESPRO READY".

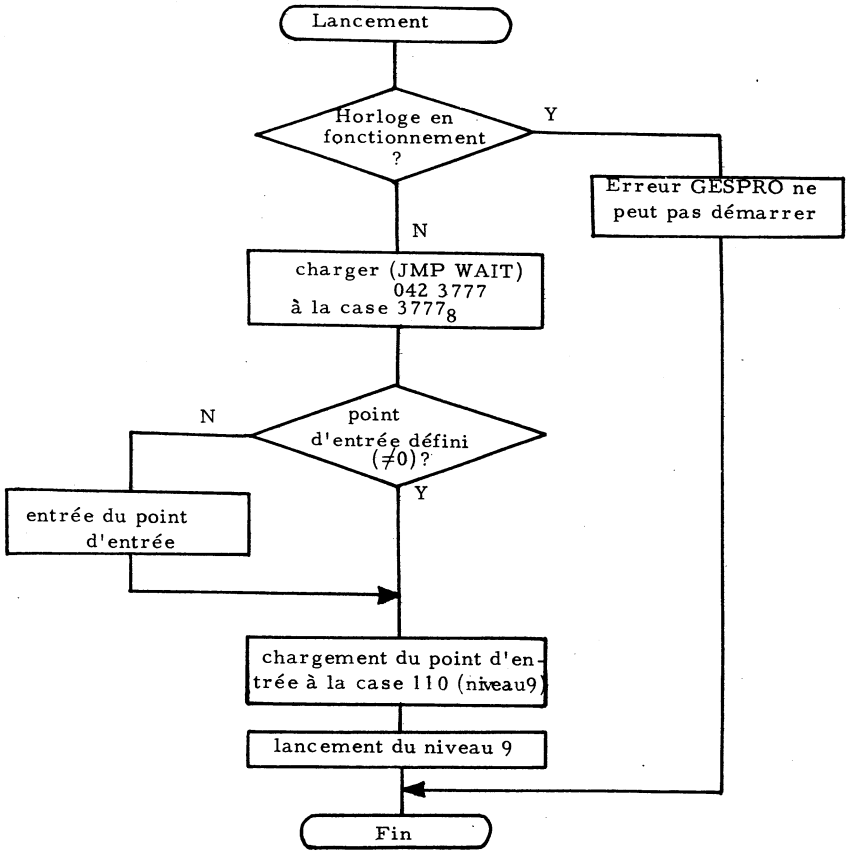


Fig. II. 46 : Organigramme du sous-programme de démarrage du processeur : GESPRO

VII. TESTS :

1. Introduction

Après conception et réalisation du système de développement, une série de tests doit nous permettre : d'une part :

- de "debugger" les programmes, au niveau instruction et fonction d'éléments d'instructions ; mais aussi dans le but
- de tester si toutes les éventualités ont été prises en compte en ce qui concerne :
  - . le format des instructions
  - . la détection d'erreurs

2. Mise en oeuvre des tests :

Une série de programmes de tests en assembleur GESPRO ont été écrits pour tester l'exécutabilité de l'assemblage principalement en fonction de :

- la longueur des programmes (problèmes d'adressage)
- la macrostructure des programmes (fonctions)
- la microstructure des programmes (instructions)
- les performances de la détection d'erreurs du point de vue syntaxique et de la sémantique.

3. Conclusion

Les tests se sont révélés très utiles, un certain nombre de points ont pu être corrigés et complétés.

CHAPITRE III

PERFORMANCES DU SYSTEME DE DEVELOPPEMENT

I. INTRODUCTION

Après avoir testé les propriétés générales du système, il s'agit maintenant d'évaluer ses performances en temps d'exécution et occupation de place en mémoire centrale et mémoire de masse, en fonction principalement de la longueur des programmes.

II. EVALUATION DES PERFORMANCES

Les performances du système de développement sont caractérisées principalement par le temps d'exécution de l'assemblage par comparaison avec d'autres assembleurs :

- l'ancien assembleur GESPRO pour évaluer le taux d'amélioration, mais aussi avec un assembleur performant
- l'assembleur de NORD (MAC).

Le temps d'exécution du chargeur va aussi faire l'objet d'une mesure de temps d'exécution mais ne sera pas comparé à d'autres vu son caractère spécifique.

1. Etude comparative des temps d'assemblage

La figure III. 1 donne le temps d'exécution d'assemblage en fonction de la longueur du programme à assembler et sans impression du nouveau et ancien assembleur GESPRO et de l'assembleur de NORD. Par rapport à l'ancien assembleur, le nouveau est exactement 100 fois plus performant en temps dans la plage d'utilisation très restreinte du premier.

Par rapport à l'assembleur NORD, il est légèrement plus performant pour des programmes inférieurs à 3, 4 K mots d'objet relogeable. Cependant pour des programmes supérieurs à 3, 4 K mots, l'assembleur NORD est un peu plus performant et d'autant plus que la longueur des programmes augmente. Ceci est dû au fait que la structure de l'assembleur NORD est telle, lorsque les symboles deviennent définis, un chainage permet de donner rapidement une valeur à tous les indéfinis relatifs à ce symbole. Il est à noter que l'assembleur GESPRO travaille sur des mots de 24 bits par l'intermédiaire de doubles mots NORD.

2. Occupation de place sur mémoire de masse par le relogeable

La figure III. 2. donne l'occupation de place en mémoire de masse par le relogeable en fonction du source, pour les programmes GESPRO et ceux de NORD.

Pour les programmes objet inférieurs à 1 K mots l'assembleur NORD est 4 fois plus performant que l'assembleur GESPRO. Pour les programmes supérieurs à 1 K mots il n'y a plus qu'un facteur 2 entre les 2 assembleurs. Quant au rapport objet/source il est de 0, 5 pour notre assembleur tandis qu'il est de 0, 25 pour celui de NORD. Ce qui signifie que la structure du relogeable de NORD est telle qu'il occupe 2 fois moins de place que le relogeable GESPRO.

Toutefois dans ce raisonnement on ne tient pas compte du fait qu'une instruction GESPRO de 24 bits est équivalente à deux instructions NORD de 16 bits du point de vue occupation de place.

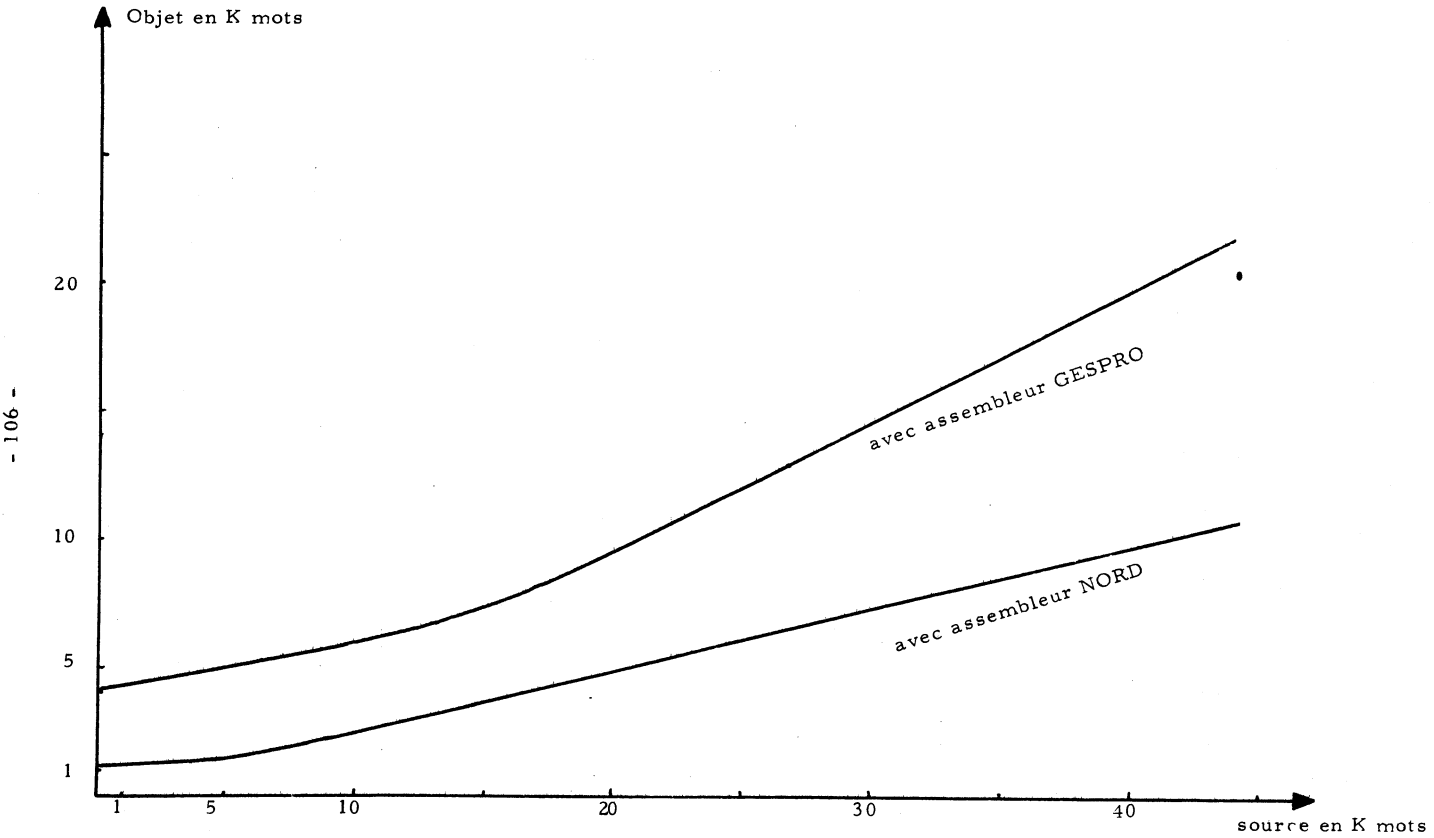


Fig. III. 2 : Rapport d'occupation de place sur mémoire de masse entre source et objet pour les programmes GESPRO et NORD

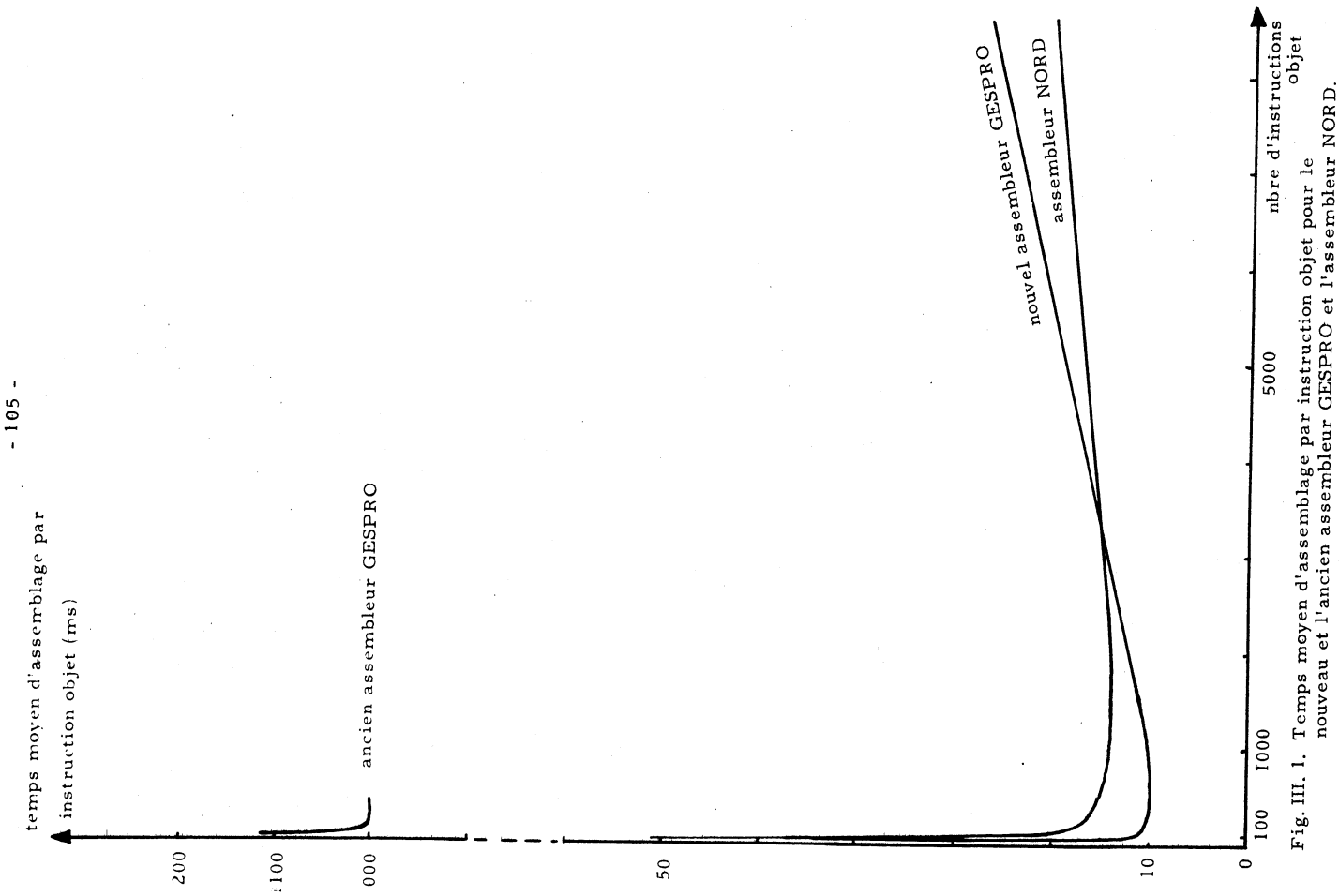


Fig. III. 1. Temps moyen d'assemblage par instruction objet pour le nouveau et l'ancien assembleur GESPRO et l'assembleur NORD.

3. Mesure du temps de chargement

La figure III.3 représente le temps de chargement de la mémoire de GESPRO, c'est à dire le temps d'exécution de l'édition de liens et du chargement en mémoire GESPRO, à partir du relogeable.

La grande valeur du temps d'exécution pour les petits nombres d'instruction est représentative du temps d'ouverture et de lecture des fichiers. Le léger accroissement du temps d'exécution est proportionnel au nombre d'instructions à charger, représentatif de l'accroissement du temps de recherche dans les tables.

III. CONCLUSION

L'assembleur comparé à celui de NORD présente des performances équivalentes, donc très acceptables.

Les performances en temps d'exécution que l'on s'était fixé par rapport à l'ancien assembleur ont été très largement atteintes.

Son utilisation, par la reprise et la mise au point du logiciel existant, s'est d'autre part révélée tout à fait satisfaisante.

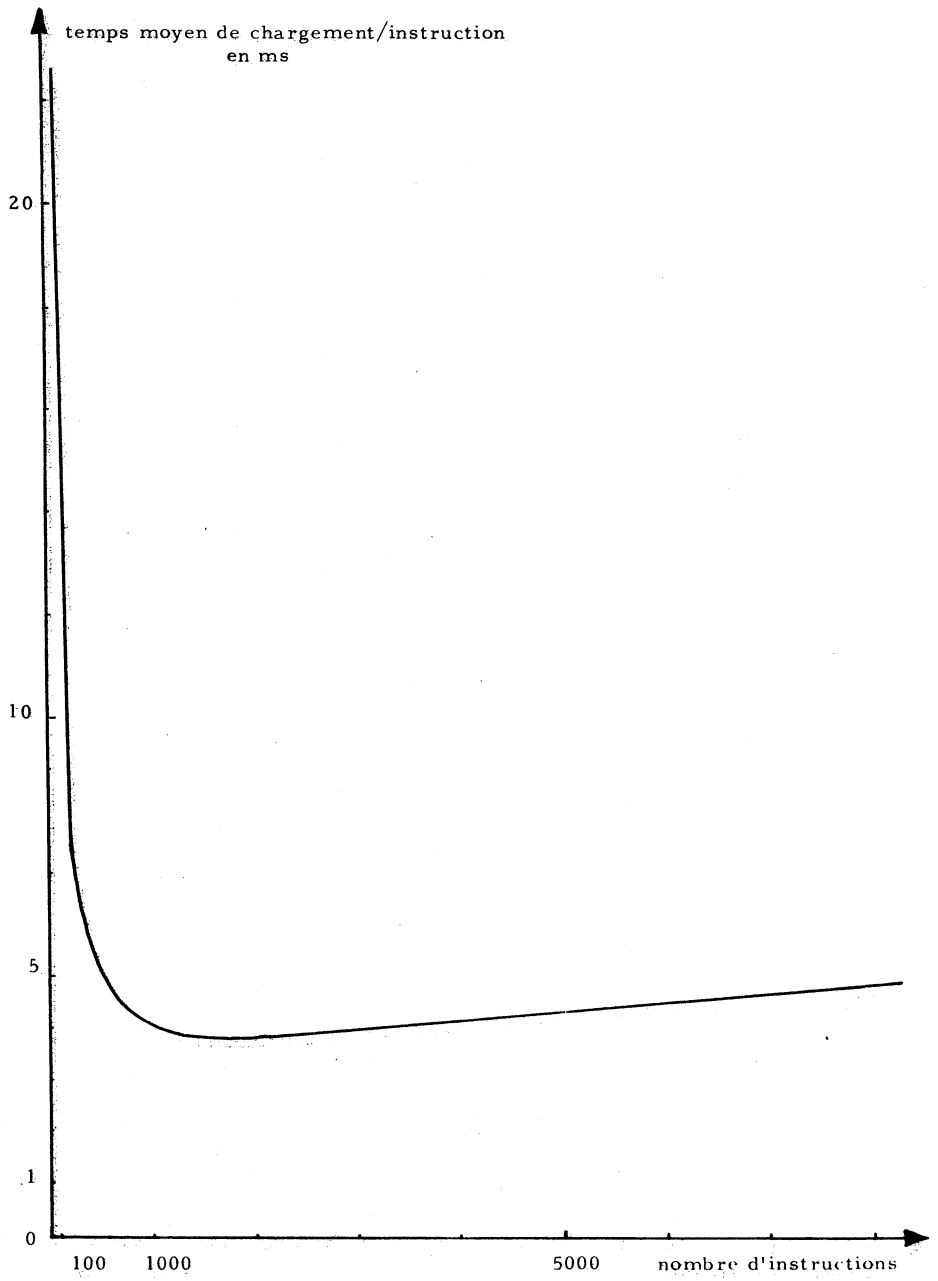


Fig. III. 3. : temps moyen de chargement de la mémoire de GESPRO, à partir du relogeable, en fonction de la longueur du programme à charger.

CONCLUSIONS

Le système de développement ainsi réalisé permet, de par sa structure de définition d'instructions, d'être généralisable à tout microordinateur et même à certains miniordinateurs au prix de légères modifications ou additions au niveau de l'assembleur.

Le chargeur dans chaque cas devra être réécrit.

Ce système de développement qui a été conçu pour GESPRO dans le cadre de l'expérience Hypérons servira prochainement dans le cadre d'une expérience de physicue faisant intervenir plusieurs microordinateurs du type de GESPRO.

Toute la souplesse de ce système de développement pourra être mise à profit.

ANNEXE A

2 DISC SPECIFICATIONS

2.1 Data Format

Word format:	16 data bits per word
Sector format:	128 data words per sector followed by a 16 bit cyclic redundancy check word.

2.2 Capacity

Capacity per sector:	128 words
Capacity per track:	3072 words
Capacity per cylinder:	12288 words
Capacity per disc:	**2506752 words**
Capacity per unit:	**5013504 words**

2.3 Access Time

Access time is defined as the sum of the cylinder positioning time and latency time.

Maximum positioning time:	70 ms
Cylinder to Cylinder positioning time:	7 ms
Average positioning time:	35 ms
Maximum latency (1 revolution):	25,5 ms
Average latency ($\frac{1}{2}$ revolution):	12,5 ms
Max access (max. pos + max. latency):	95,5 ms
Average access (av. pos + av. latency):	47,5 ms

2.4 Transfer Rate

Bits:	2,5 MHz	=	400 ns/bit
Words:	156 KHz	=	6,4 µs/word
Sectors:		=	1,06 ms/sector

2.5 Power Requirement

Voltage:	220V AC ± 10% 49 - 50,5 Hz
Current:	3,5A
Power:	600 watt

Impression du contenu du fichier de définitions d'instructions en format identique à celui de la rentrée des instructions dans le fichier à partir de la console.

Date 0 2 8 3 3 1978

Nom { ---FILE : INSD:DATA

Entête { 003462 004000 001250
INSD 001247 000564

```
1250   ACTLV    8,11,5          '147,0,E(1)
1262   ADD      6,2,16          '67,C(1),A(1)
1274   ADDI     7,1,1,15          '146,C(1),C(2),A(1)
1311   ADIM     7,1,16,24          '105,C(1),0,D(1)
1326   AND      7,1,16          '110,C(1),A(1)
1340   ANDI     7,1,1,15          '143,C(1),C(2),A(1)
1355   ANDIM    4,1,3,16,24           '10,C(1),4,0,D(1)
1375   CALL     8,16          '63,A(1)
1404   CLAL     8,16          '143,0
1413   CLLV     8,11,5          '146,0,E(1)
1425   COPY     4,2,2,16          '16,C(1),C(2),A(1)
1442   CP       6,2,16          '75,C(1),0
1454   CPONE    8,16,24          '73,A(1),D(2)
1466   CPZRO    8,16,24          '77,A(1),D(2)
1500   DACAM    4,5,3,3,5,4           E(1),E(2),E(3),E(4),E(5),E(6)
1523   DATHL    8,16          E(1),E(2)
1532   DCM      8,16          '60,A(1)
1541   DCMI     8,1,15          '305,C(1),A(1)
1553   DRJ      4,2,2,16          '1,C(1),2,A(1)
1570   DSKLV    8,11,5          '145,0,E(1)
1602   ENIMA    8,16,8,16,8,16,8,16,8,16,8,16          '252,A(1),E(2),E(3),51,A(4),51,A(5),51,A(6)
1647   ENTER    24          '0
1653   GIVUP    8,16          '167,2048
1662   IMAD     8,16,24,8,16          '251,A(1),A(2),E(3),E(4)
1702   IMADD    7,1,16,8,16          '105,C(1),0,E(1),E(2)
1722   IMAND    4,1,3,16,8,16          '10,C(1),4,0,E(1),E(2)
1745   IML      6,2,16,8,16          '40,C(1),0,E(1),E(2)
1765   IMOR     4,1,3,16,8,16          '10,C(1),5,0,E(1),E(2)
2010   IMSUB    4,1,3,16,8,16          '10,C(1),7,0,E(1),E(2)
2033   IMXOR    4,1,3,16,8,16          '10,C(1),6,0,E(1),E(2)
2056   INM      8,16,8,16,8,16,8,16          '71,A(1),34,A(2),34,A(3),34,A(4)
2107   INDM     8,16,8,16,8,16,8,16          '71,A(1),51,A(2),51,A(3),51,A(4)
```

↳ adresses en zône enregistrement

ANNEXE B

IMPRESSION DU CONTENU REEL DU FICHIER DE DEFINITIONS D'INSTRUCTIONS.

```
Nom du fichier {---FILE : INSD:DATA  -----
Entête         { 000000 000707 002030 000524
                 000003 INSD   000523 000060

                 000010 ADD    000524
                 000014 ADDI   000536
                 000020 AND    000553
                 000024 LD     000565
                 000030 LDI    000577
                 000034 MOVE   000614
                 000040 ST     000631
                 000044 STI    000643
                 000050 SUB    000660
                 000054 SUBI   000672
Index

                 000524 000001 000001 000067 000006 000001
                 000532 000002 000017 000001 000020 000001
                 000540 000146 000007 000016 000001 000015
                 000546 000002 000001 000017 000001 000017
                 000554 000001 000110 000007 000016 000012
                 000562 000017 000001 000020 000012 000001
                 000570 000006 000016 000015 000001 000002
                 000576 000020 000001 000001 000140 000007
                 000604 000001 000001 000016 000002 000001
                 000612 000001 000017 000015 000001 000210
                 000620 000017 000001 000020 000020 000002
                 000626 000017 000003 000010 000012 000001
                 000634 000006 000016 000001 000002 000017
                 000642 000020 000015 000001 000141 000007
                 000650 000001 000001 000016 000002 000001
                 000656 000035 000017 000012 000002 000001
                 000664 000016 000001 000002 000017 000001
                 000672 000015 000001 000147 000007 000016
                 000700 000001 000016 000002 000001 000017
                 000706 000017 001354 131640 000011 136556
Zône enregistrement (définitions d'instructions)
```

adresses de repérage sur listing

Information contenue dans le fichier

ANNEXE C

Listing à l'assemblage

```
23 41  6      9    6   1978
-------------------------------

SOURCE-FILE NAME ############## HUOT1:SYMB

                                      BEG   START
                                      DEF    BUF
                                      REF    MNBV
00001      322 000077      START   LD,D   '77
                           TREW    SET   '34
00002      320 000000              LD,A   START
00003      322 000034              LD,D    TREW
00004      106 000000              ST,D   BUF
00005      106 000000              ST,D   MNBV
00006      000 007777              DATA   '7777,5555,888888
00007      000 012663
00010      015 110070
00014      000 000000      *+4
00015      000 000001      BUF     DATA   1,0
00016      000 000000
                           $NAME   END


PRINT OBJECT YES OR NO ?
N

 CONTINUE  YES OR NO ?
```


Listing des tables

ENTETE DU FICHIER

```
 NAME
000024      000060  ---> bornes table objet
002000      002017  ---> bornes table de translation
004000      004000  ---> bornes table des références extérieures
004100      004105  ---> bornes table des définitions extérieures
005000      005004  ---> bornes table des directives (IF)
006000      006044  ---> bornes table des symboles + valeurs
000000  ------------> nombre d'erreurs
  START     000001  ---> point d'entrée
```

DICTIONNAIRE DE TRANSLATION

```
000002   000000   000020   000020   000001
000004   074156   000020   000020   000015
000005   000001   000020   000020   000000
```

TABLE DES REFERENCES EXTERIEURES

```
000001   MNBV      000000
```

TABLE DES DEFINITIONS EXTERIEURES

```
BUF        000015
```

TABLE DES SYMBOLES ET LEUR VALEUR

```
A          000000
B          000001
BUF        000015
D          000002
E          000003
ID         000000
IE         000001
START      000001
TREW       000034
```

ANNEXE D

Listing au chargement

```
******ERREUR SYMBOLE : MNBV
NON DEFINI DANS : HUOT1:DATA

 0  38  54       10     6    1978
-----------------------------------
```

```
SOURCE-FILE NAME ************: HUOT1:SYMB


                                    BEG   START
                                    DEF    BUF
                                    REF    MNBV
000000          322   000077   START  LD,D   '77
                                 TREW    SET   '34
000001          320   000000          LD,A   START
000002          322   000034          LD,D    TREW
000003          106   000014          ST,D   BUF
000004          106   000000          ST,D   MNBV
000005          000   007777          DATA   '7777,5555,888888
000006          000   012663
000007          015   110070
000010          000   000000   *+4
000011          000   000000
000012          000   000000
000013          000   000000
000014          000   000001   BUF     DATA   1,0
000015          000   000000
                                   $NAME   END

        SYMBOLES ET LEUR VALEUR EN MEMOIRE GESPRO


MODULE OBJET :HUOT1:DATA

      A          000000
      B          000001
      BUF        000014
      D          000002
      E          000003
      ID         000000
      IE         000001
      START      000000
      TREW       000034
```


BIBLIOGRAPHIE

DANOVAN, J. J. : Systems Programming

BARRON, D. W. : Assemblers and loaders

KATZEN, H., Jr. : Advanced Programming

ROSEN, S. : Programming systems and langages

RODNAY, Z. : Microprocessors from chips to systems

1. Contribution à l'étude des systèmes informatiques en temps réel en physique des particules élémentaires - thèse de doctorat ès sciences - J. M. MEYER (1977) - Université du Haut-Rhin
2. Contribution à l'étude et à la réalisation d'un processeur micro-programmable, ultra-rapide, destiné à la gestion des tâches CAMAC - thèse de doctorat de 3è cycle - C. BOULIN (1976) Université Louis Pasteur Strasbourg I
3. CAMAC - A modular instrumentation system for data handling (description and specification) - EURATOM Report 4100 and 4600
4. NORD-10 - Users manual - NORSK DATA A.S. -
5. SINTRAN III - Users guide - NORSK DATA A.S. -
6. File system - NORSK DATA A.S. -
7. FLORES, I., MADPIS, G. : Average binary search length for dense ordered lists
8. QUITTNER, P., Problems, Programs, Processing results, Software techniques for sciences techniques programs.

REMERCIEMENTS

Je remercie Monsieur le Professeur G.SUTTER de m'avoir fait l'honneur d'accepter la présidence du jury de thèse.

Je prie Monsieur M.CROISSIAUX de bien vouloir trouver ici l'expression de ma profonde gratitude pour l'accueil qu'il m'a réservé dans son laboratoire.

Je tiens à remercier Monsieur le Professeur G.METZGER qui par son enseignement m'a conduit à découvrir un domaine de recherches passionnant.

Monsieur J.M.MEYER a bien voulu diriger cette thèse, je lui en suis profondément reconnaissant.

Mes remerciements les plus sincères vont également à Messieurs J.LECOQ et M.PERRIN pour leur précieuse collaboration à l'accomplissement de ce travail.

Je remercie Messieurs J.D.BERST et Y.CHATELUS pour l'aide et les encouragements qu'ils m'ont produigés.

Je tiens enfin à remercier Madame GOETZ pour la réalisation matérielle de cette thèse.